



Università degli Studi di Bologna  
Scuola di Ingegneria

# Corso di Reti di Calcolatori T

***TCP/IP: protocolli e scenari di uso***

**Antonio Corradi**  
Anno Accademico 2022/2023

# SUITE TCP/IP E INTERNET

---

Gli standard possono nascere da comitati o anche dal basso da esigenze di uso e con obiettivo di realizzazione immediata

Internet nasce dalla idea di potere interconnettere tutte le reti in una unica globalità (il migliore dei mondi possibili)

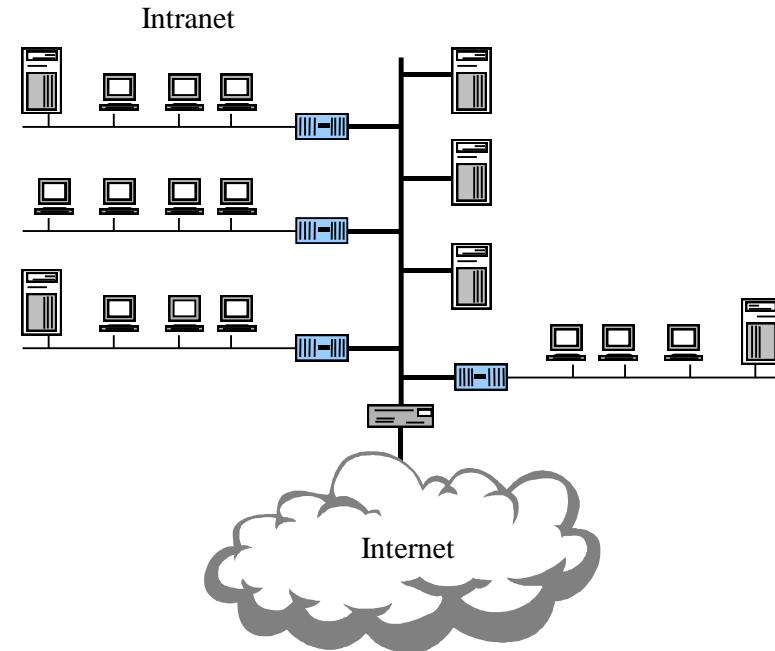
**SISTEMA GLOBALE** a nessun costo e per tutti

protocolli liberi, aperti, a nessun costo

Intranet come insieme di reti aziendali che adottano protocolli standard IETF

**SISTEMA di RETI** per scopi aziendali con problemi di sicurezza, di accesso, di controllo, di accounting, ...

Protocolli a basso costo per la comunicazione con il sistema globale



# I DUE LIVELLI TCP/IP

## TCP - Transmission Control Protocol

- flusso di byte bidirezionale a canale virtuale best effort, dati non duplicati, affidabili, con controllo di flusso

livello TX

## UDP User Datagram Protocol

- scambio di messaggi end-2-end

livello TX

## IP Internet Protocol (Routing)

- scambio di datagrammi senza garanzia di consegna tra vicini

livello di RETE

## ICMP Internet Control Message Protocol

- scambio messaggi di controllo

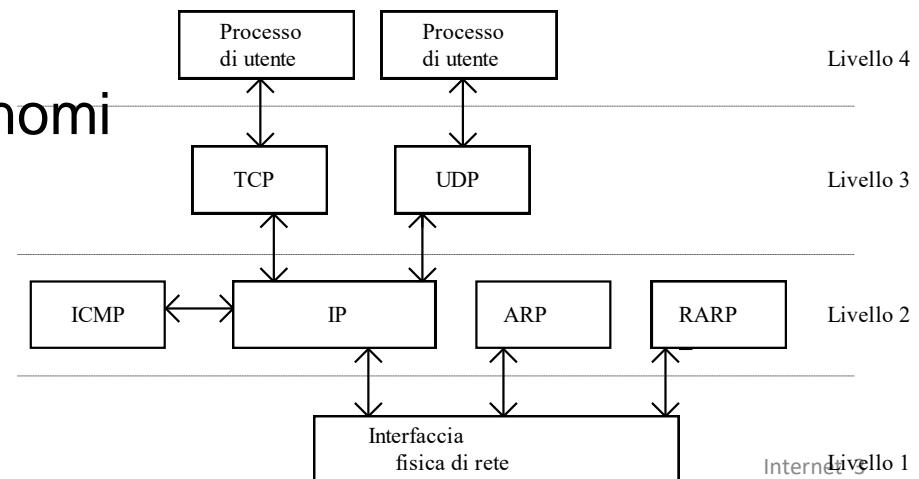
gestione RETE

## ARP e RARP Protocol

- Interazione con livello fisico e nomi

## STACK di protocolli

a basso overhead e best effort



# ADDRESS RESOLUTION PROTOCOL

---

I protocolli devono tenere conto anche della visione verticale

Nell'invio di datagrammi, si devono risolvere gli indirizzi per il livello **data link** (il livello di rete deve fornire qualche modo per farlo)

Due macchine che comunicano hanno sia

indirizzi di IP:      la,      lb      (RETE)

indirizzi fisici:      Fa,      Fb      (DATA LINK)

- La risoluzione dell'indirizzo fisico potrebbe avvenire con mappaggio diretto, estraendo il nome fisico dall'indirizzo IP, ma violerebbe la indipendenza dei livelli tra loro, inoltre i nomi sono svincolati

In reti standard si richiede un protocollo dinamico che permetta di ritrovare il nome fisico Internet (ossia di data link) dal nome IP

Si sono definiti protocolli per supportare questi SISTEMI di NOMI:  
i protocolli sono ARP e RARP

# ARP PROTOCOL

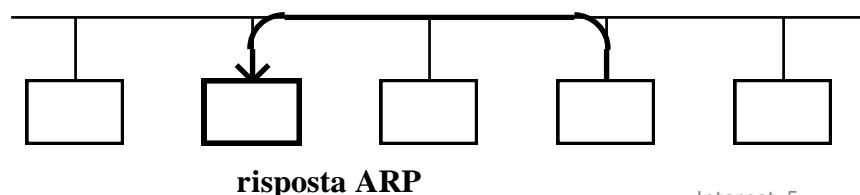
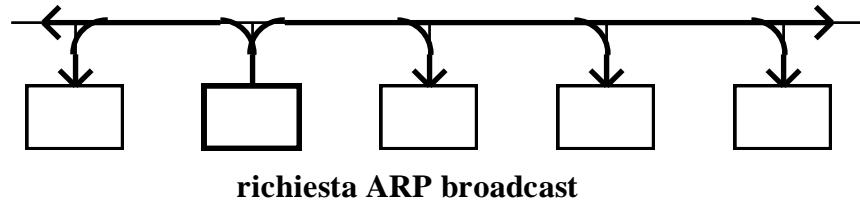
**ARP è il protocollo per la ricerca dell'indirizzo fisico di un nodo partendo dal suo indirizzo IP**

ARP protocollo solo locale (basato su broadcast) semplice ed efficiente

- Invio di un pacchetto broadcast in cui si chiede l'indirizzo fisico (Fa) corrispondente ad indirizzo IP - Quale Fa per questo IP?
- tutti gli host ricevono tale pacchetto e solo quello che riconosce il suo indirizzo IP risponde con il proprio indirizzo fisico

I protocolli devono tenere conto dei costi nel progetto, vedi comando per visualizzare la cache

`arp -opzioni`



# ARP PROTOCOL: CACHE

---

## I protocolli devono tenere conto dei costi di esercizio

Il protocollo di **broadcast locale** non viene attivato per ogni pacchetto

- Altrimenti per ogni datagramma ne manderemmo altri per la parte di supporto ai nomi
- Utilizzo di una **memoria cache** per mantenere le associazioni {indirizzo IP - indirizzo fisico} già usate
- La **cache** viene consultata prima di comunicare in ARP

## Se si attua il broadcast si cerca di usarlo al meglio...

- *l'associazione relativa alla macchina richiedente memorizzata anche dalla macchina che risponde ad ARP*
- *ogni richiesta broadcast viene memorizzata da tutti*
- *ogni nuova macchina al collegamento invia sulla rete locale un broadcast con la propria coppia {indirizzo fisico - indirizzo IP}*

# ARP PROTOCOL: RUOLI

---

ARP distingue due ruoli nel protocollo, che ogni nodo realizza

- **uno attivo** richiede l'indirizzo fisico per un pacchetto
- **uno passivo** risponde alle richieste di altri

## Attivo

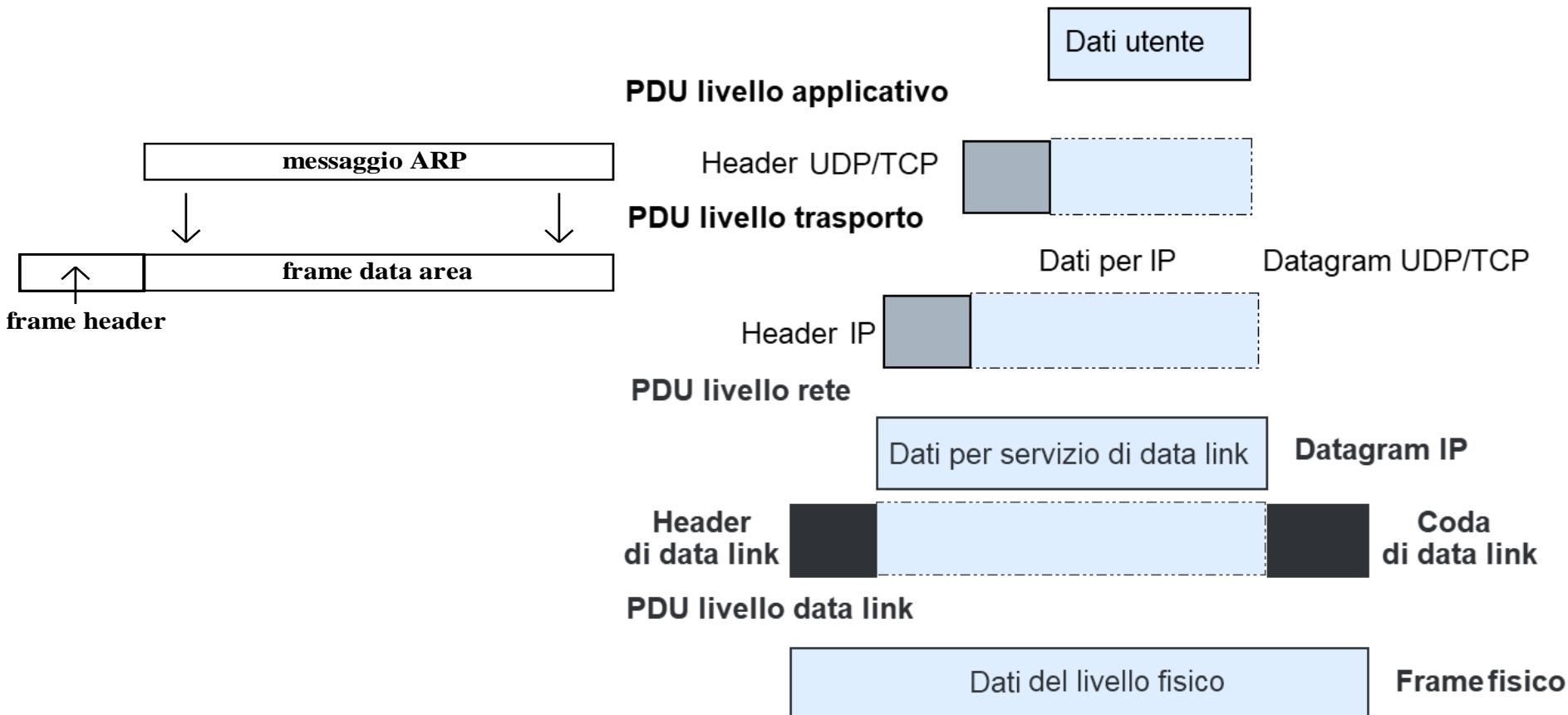
- **esamina la cache per risolvere indirizzo IP localmente**, altrimenti esegue una richiesta ARP broadcast
- ***la gestione della richiesta broadcast deve prevedere di non ricevere risposta o riceverla con ritardo***

## Passivo

- **risponde alle richieste di altri** (come server) processando il pacchetto
- **estraendo sia indirizzo IP sia il fisico** per pacchetto ARP
- **Se richiesta del proprio indirizzo  $\Rightarrow$  invio risposta**

**Un messaggio ARP encapsulato in frame fisici e reinviato al richiedente**

# ARP: REALIZZAZIONE



Ogni PDU formato ad un livello diventa il SDU per il livello inferiore

# DATA LINK: ETHERNET

ETHERNET, standard di fatto a livello MAC (Medium Access Control)  
*campi in byte*

<b>PREAMBOLO</b>	7	10101010
<b>delimitatore di inizio frame (Start Frame Delimiter)</b>	1	11010101
<b>DESTINATION address</b>	6	
<b>SOURCE address</b>	6	
<b>type (id protocollo ARP/RARP ...)</b>	2	
<b>DATA</b>	46.. 1500	
<b>controllo di fine frame (Frame Check Sequence)</b>	4	

**Formato di un frame con indirizzi a 6 byte o 48 bit**  
con dati di lunghezza variabile da 46 a 1500 ottetti

# DATA LINK: ETHERNET

Anche gli altri livelli MAC introducono forme analoghe per i frame corretti

<i>campi in byte</i>	
PREAMBOLO	7 10101010
delimitatore di inizio frame (Start Frame Delimiter)	1 11010101
DESTINATION address	2/6
SOURCE address	2/6
type (id protocollo ARP/RARP ...)	2
DATA . . .	46.. 1500
controllo di fine frame (Frame Check Sequence)	4

In genere:

- indirizzi a **48 bit** per il nodo mittente e destinatario
- si introducono sia preamboli, sia delimitatori finali
- controllo del frame attuato con **controllo CRC**

**In un frame, per l'invio di 1 solo byte (applicativo oltre il trasporto)**

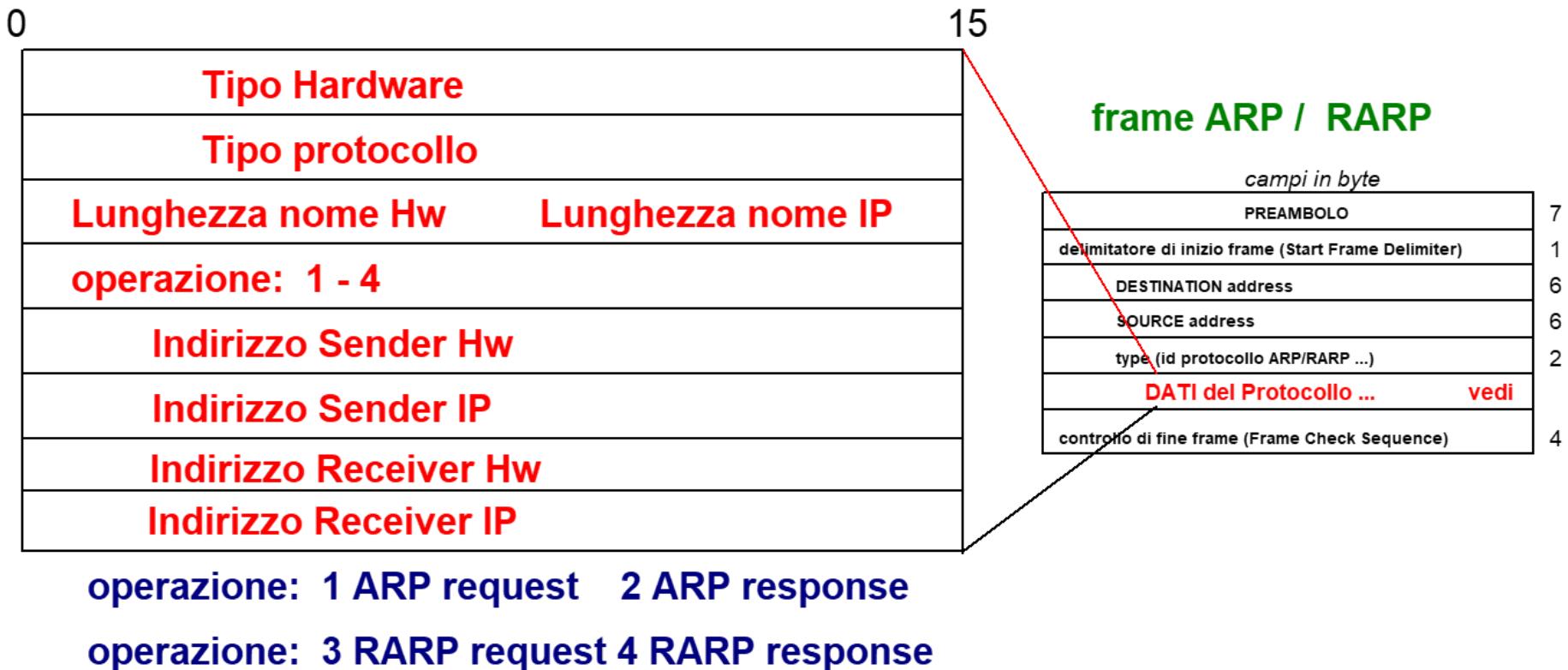
**con 1 byte applicativo** ⇒ **overhead 46 byte**

**20 IP e 20 TCP/UDP, 5 riempimento**

# FORMATO HEADER ARP / RARP

Si possono considerare i due protocolli insieme

Le informazioni di protocollo sono inserite in un frame di livello data link



# PROTOCOLLO RARP

---

**Protocollo Reverse Address Resolution Protocol** attua la ricerca di indirizzo IP per i nodi che non conoscono il proprio indirizzo IP e conoscono solo il proprio indirizzo MAC (Data Link)

- In genere, **l'indirizzo IP si mantiene sul disco**, e lì viene trovato dal SO
- Per le macchine **diskless**, l'indirizzo IP **viene ottenuto richiedendolo ad un server di rete** che contiene tutti gli indirizzi IP di una rete

**RARP usa la rete fisica** e gestisce la ritrasmissione e la perdita di messaggi tramite **alcuni server RARP**

**Si prevedono più server per ogni LAN** per rispondere ai clienti anche in caso di guasto

Cliente - usa broadcast di data link per raggiungere il server RARP  
e se non si ottiene risposta? ritrasmissione

Servitore - invia la risposta a chi ne ha fatto richiesta

# SERVER RARP

---

Se sono previsti, spesso ci sono **server multipli per RARP**

## **Modello a server attivi**

- Troppi server sovraccaricano il sistema se cercano di rispondere contemporaneamente alla richiesta

## **Modello a server attivi/passivi**

- soluzioni possibili con gerarchia di server

## **Come si può evitare la interferenza dei server?**

- **Modello dinamico** con server differenziati in ascolto: il server primario è il solo a rispondere; gli altri server rispondono solo se arriva una seconda richiesta RARP (anche in gerarchia)
- **Modello statico** con server differenziati e ritardi diversi: si prevede che il server primario risponda immediatamente, gli altri con un ritardo calcolato random, con una bassa probabilità di risposta simultanea

**ARP protocollo molto usato, RARP deprecato**

# DYNAMIC Host CONFIGURATION PROTOCOL

---

## Dynamic Host Configuration Protocol o DHCP (rfc 2131)

protocollo per la attribuzione dinamica di indirizzi IP per ottenere una configurazione dinamica e con risparmio rispetto ad IP statici

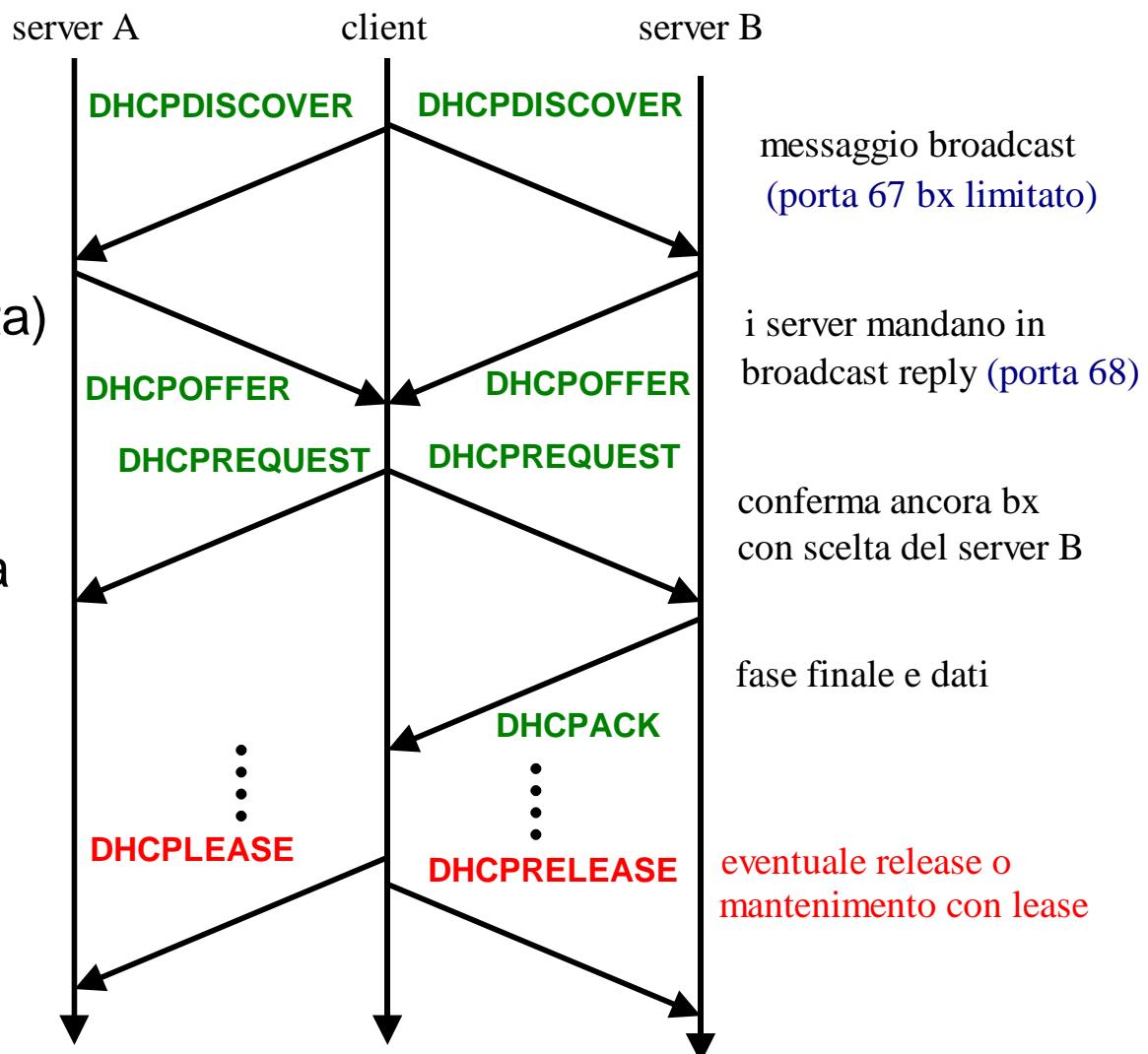
- I provider devono risparmiare gli IP ed assegnarli secondo le richieste ai clienti che non ottengono IP permanenti ma solo assegnati su bisogno

**Si basa su due ruoli distinti: clienti e servitori con protocollo di offerta tipo asta (bidding) a più fasi, con iniziativa cliente**

- **broadcast** della **richiesta** di discovery (richiesta di ingresso)
- **offerte** da parte dei servitori (con parametri di scelta)
- **scelta** di una offerta (in broadcast)
- **conferma** della offerta
- messaggi prima della scadenza (**lease**)
- rilascio dell'offerta (**release**)

# BIDDING PROTOCOL A 4 FASI

1. broadcast della **richiesta** di discovery
2. **offerte** dei servitori (con parametri di scelta) anche non in bx
3. **scelta** di una offerta (anche in broadcast)
4. **conferma** della offerta
  - rilascio dell'offerta (**release**)
  - messaggi di conferma prima della scadenza (**lease**)



# FASI DHCP: BIDDING PROTOCOL

---

## Protocollo di bid fuori linea in diverse fasi (almeno 4)

- **discover** come broadcast della richiesta ai server
- più **offer** dei servitori al cliente (quante sono attese? Per quanto?)
- **scelta di una offerta** con una request (da notificare al/ai server)
- **conferma e offerta** con ack e dati ulteriori

## Per la fase di rilascio

- **rilascio dell'offerta in modo esplicito o implicito** a tempo (messaggio di release per lasciare la risorsa)
- **mantenimento dell'indirizzo** con messaggio di lease prima della scadenza della attribuzione

# DHCP PROTOCOL

---

Le moderne organizzazioni tendono a usare il protocollo DHCP

- per gestire **multi host di una organizzazione**
- (evitando **set-up manuale o statico** e per ragioni di sicurezza)

**Al contratto viene associata una durata: se durante l'intervallo non si usa, il server può riassegnare l'indirizzo**

- (con un **soft-state**) consente di riusare la attribuzione dopo un certo tempo senza un uso effettivo

**Il lease permette di confermare l'uso, senza rieseguire il protocollo**

- **DHCP** permette l'attribuzione di tutta una serie di parametri di gestione: maschera di rete, sottorete, diritti, ecc. ecc.
- **uso di broadcast** (protocollo locale) con più server che memorizzano tutte le informazioni relative ai clienti gestiti
- **molto utilizzato per host mobili**, sistemi wireless, dispositivi limitati da fare entrare in una rete di una organizzazione

# NETWORK ADDRESS TRANSLATION NAT

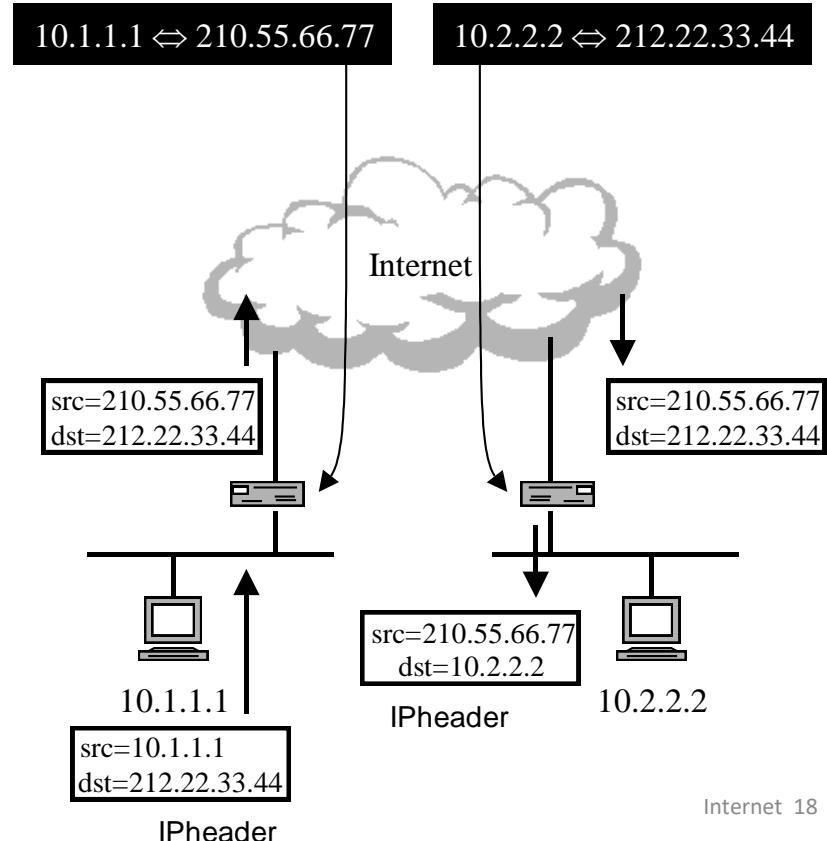
NAT protocol usato per traslare indirizzi intranet privati in indirizzi IP globali in rete aperta (uso di indirizzi riconosciuti da IETF)

per superare il problema di indirizzi privati intranet che debbano arrivare in reti Internet

- Uso di router NAT che attuino le traslazioni mantenendo tabelle apposite

NAT permette di riferire da una intranet con indirizzi interni indirizzi esterni di Internet usando tabelle di corrispondenza sul router NAT

Esistono varie forme di NAT, a volte con cascate di router



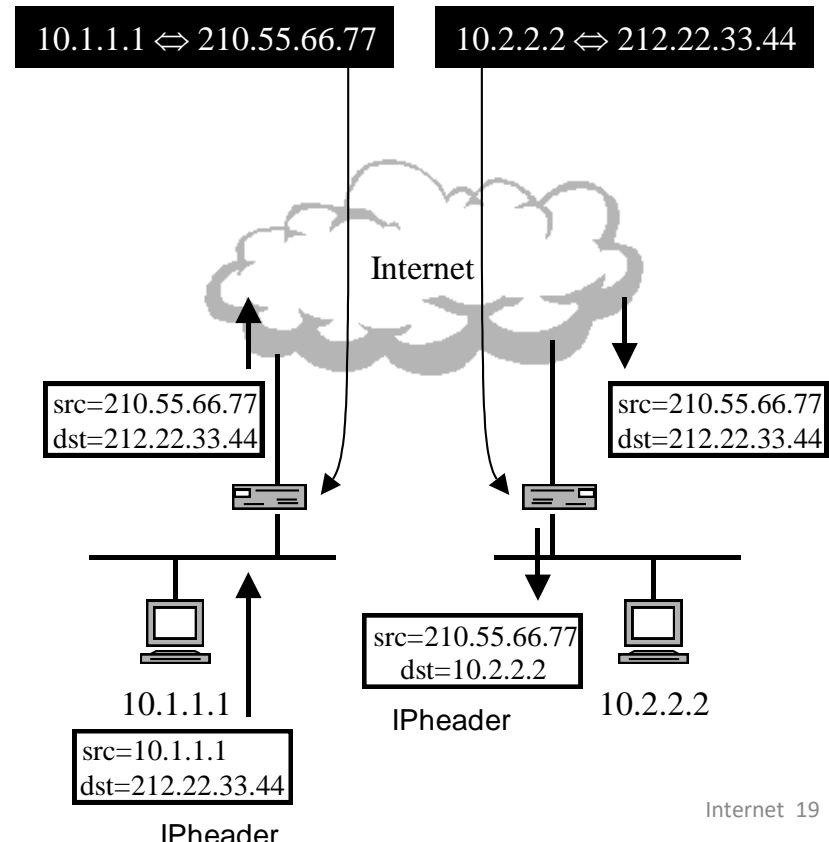
# NETWORK ADDRESS TRANSLATION NAT

NAT protocol viene usato in molte accezioni, a parte quella semplice vista (uso di indirizzi locali non autorizzati e non riconosciuti da IETF)

In generale si dovrebbero istruire i DNS per fare delle *traslazioni statiche o dinamiche* in modo da arrivare a situazioni anche molto articolate

- Non solo uso di router NAT traslando con tabelle apposite  
Ma coordinamento di altri sistemi di nomi per la consistenza

NAT a livello network presenta problemi per applicazioni che usano nomi IP a livello applicativo



# NAT PROTOCOL

---

**NAT protocol viene supportato da router e proxy** che devono traslare ogni comparsa di indirizzi privati in indirizzi esterni e validi

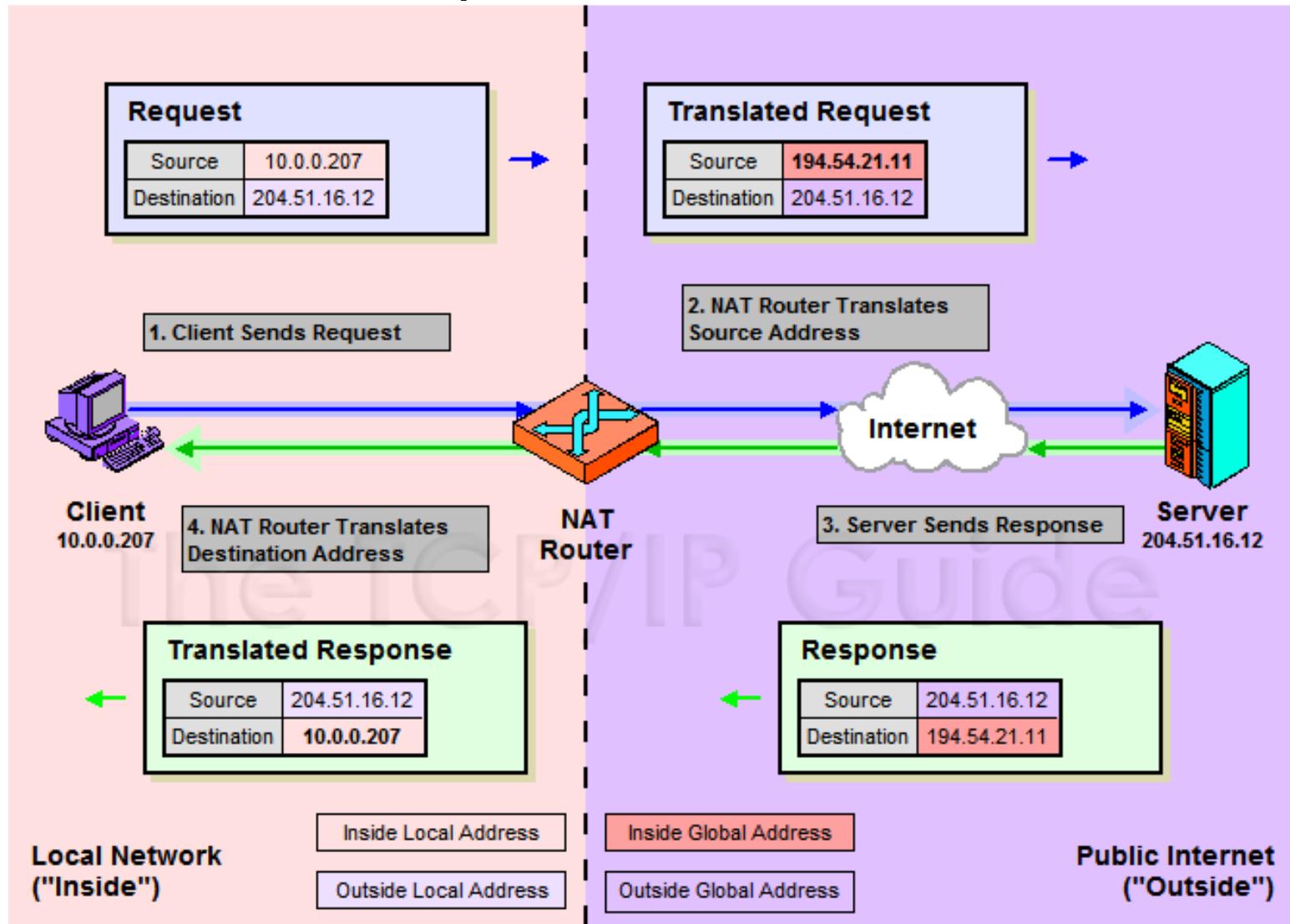
- In genere **un indirizzo interno viene mappato in un indirizzo esterno**: questo richiede tabelle di corrispondenza univoche (stesso numero di indirizzi interni ed esterni)
- Ogni datagramma **deve essere trattato, identificando indirizzi e traslando nei due sensi**

**Si è previsto un NAT più espressivo, detto NAT a porte, per usare 1 solo indirizzo esterno per tutti gli interni, impegnando meno risorse esterne (vedi provider)**

- NAT a porte **permette di mappare molti indirizzi interni in un unico indirizzo esterno**, distinguendo i molteplici indirizzi interni attraverso un uso di porte diverse di connessione per il router, associate ai diversi indirizzi interni
- Questo **limita l'impegno di indirizzi esterni** (con molti indirizzi interni associati) e i costi di assegnamento

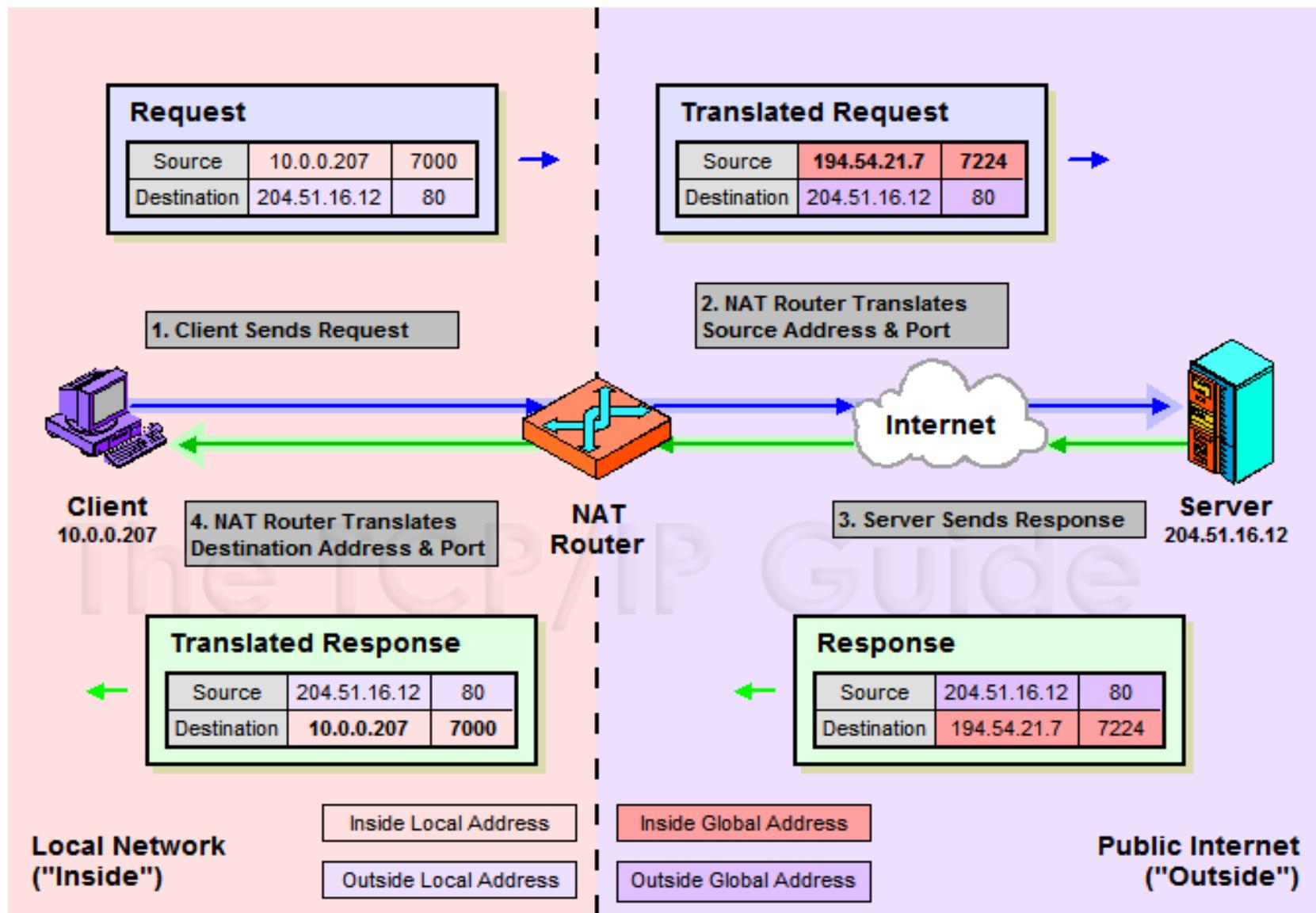
# NAT TRADIZIONALE

Il router NAT trasla da indirizzo interno ad esterno disponibile solamente tenendo traccia delle corrispondenze



# NAT A PORTE

Il router NAT si presenta all'esterno usando proprie porte molteplici



# INTERNET CONTROL MESSAGE PROTOCOL

---

**Internet Control Message Protocol (ICMP) come protocollo di gestione e controllo su IP migliorando la qualità best-effort**

- ICMP consente di inviare messaggi **di controllo o di errore**, in genere, al nodo sorgente del messaggio (e solo a questo)
- ICMP usato anche per **coordinare le entità di livello IP con azioni di gestione**

**ICMP rappresenta un mezzo per rendere note condizioni anomale a chi ha mandato datagrammi (usando IP) o azioni di gestione**

- come **puro meccanismo di eccezione** con politica di uso tutta a carico **dell'utilizzatore**

**ICMP viene attivato per segnalare al mittente di datagrammi condizioni di errore (non correzione) per avviare provvedimenti**

- nodi intermedi non informati dei problemi
- nodo sorgente può provvedere a correggere

# INTERNET CONTROL MESSAGE PROTOCOL

---

**Internet Control Message Protocol (ICMP) usa il normale routing IP per arrivare al mittente con messaggi ICMP che**

- sono imbustati in un datagramma IP
- sono soggetti alle stesse regole di routing
- non hanno priorità
- possono essere persi
- possono causare ulteriori congestioni



**Non si attua nessun supporto ad-hoc per la gestione**

Sono messaggi di **METALIVELLO** ma a basso costo

- e gli errori sugli errori? Se si perde un messaggio ICMP?
- Errori su messaggi ICMP non possono causare a loro volta messaggi ICMP altrimenti corriamo il rischio di effetti di congestione ...

# ICMP FORMATO

---

**Formato dell'header ICMP che viene inserito in un datagramma IP e smistato in modo trasparente**

- il messaggio ICMP contiene sempre l'header e i primi 64 bit dell'area dati del datagramma che ha causato il problema

0	8	16	31
TYPE	CODE	CHECKSUM	
DATA			
in caso di errore l'header del datagramma sbagliato			
...			

type

**identificatore del messaggio**

code

**informazioni di tipo messaggio**

checksum

**(16 bit) utilizzato dal relativo algoritmo**

# INFORMAZIONI TYPE ICMP

---

I campi **type** e **code** consentono di fornire informazioni ulteriori al mittente segnalando le cause del problema

**Possibili valori del campo type** (bold errori, gli altri gestione)

0	<i>Echo Reply</i>
3	<b>Destinazione irraggiungibile</b>
4	<b>Problemi di congestione (<i>source quench</i>)</b>
5	Cambio percorso ( <i>redirect</i> )
8	<i>Echo Request</i>
11	<b>Superati i limiti di tempo del datagramma</b>
12	<b>Problemi sui parametri del datagramma</b>
13	Richiesta di timestamp
14	Risposta di timestamp
15	Richiesta di Address mask
16	Risposta di Address mask

# INFORMAZIONI CODE ICMP

---

## Gli eventi segnalati al meglio

campo **CODE** ⇒ un intero dipendente dai valori di TYPE

Se il destinatario non si raggiunge

campo type vale 3 e campo code codice di errore

## Possibili valori del campo code

- 0** Rete irraggiungibile
- 1** Host irraggiungibile
- 2** Protocollo irraggiungibile
- 3** Porta irraggiungibile
- 4** Frammentazione necessaria
- 5** Errore nel percorso sorgente (source route fail)
- 6** Rete di destinazione sconosciuta

# ICMP: LIVELLO ERRORI

---

## Destination unreachable (type 3)

- Network unreachable (code 0)
- Frammentazione necessaria, ma non consentita
- Route a sorgente non esatta (source route failed)

## Source quench (type 4)    caso di congestione

- Se il buffer dedicato ai frammenti e datagrammi è esaurito, sono scartati: si invia un avvertimento al mittente

## Cicli e perdita di datagrammi (type 11)

- problemi su un datagramma singolo
- scadenza del time-to-live o del tempo di ricomposizione

# **ICMP: LIVELLO COORDINAMENTO**

---

**ICMP permette l'invio di informazioni di routing tra gateway  
echo request/reply (type 8/0) - controllo percorso**

- un host verifica la raggiungibilità di una destinazione
- Si può verificare che un host esista
- inviando un echo request (type 8)
- attendendo la ricezione di echo reply (type 0)

**address mask (type 17/18) - richiesta di maschera**

un gateway deve conoscere una sottorete

- sincronizzazione degli orologi (type 13/14) ricezione e invio del tempo fisico misurando i millisecondi
- si considera tempo di invio, di ricezione, di risposta

**redirect (type 5) - cambio percorso**

- un gateway deve cambiare la propria tabella di routing

# ICMP: STRUMENTI DI GESTIONE

---

**Comando `ping`** per stimare il RoundTrip Time o RTT

- si mandano **echo request** al nodo e attesa di **echo reply**.
- si possono **variare le dimensioni dei dati ed il numero di invii**, secondo le diverse necessità di gestione

**Comando `traceroute` (o `tracert`)** per visualizzare il percorso da un nodo fino ad un altro nodo

- si mandano **messaggi con TimeToLive crescente**
- il nodo che **riceve il datagramma con TTL=0** lo scarta e manda un messaggio ICMP al mittente
- **ogni perdita forza un messaggio ICMP** catturato dal mittente

Il mittente riceve quindi messaggi da ogni nodo del cammino e può ricostruire il cammino stesso

**assumiamo che nel frattempo non cambino le tabelle di routing  
altrimenti potremmo avere dei problemi di inconsistenza**

# UDP: USER DATAGRAM PROTOCOL

UDP protocollo di trasporto (Tx) best-effort e a basso costo

**Tx processo a processo**

**Rete nodo a nodo**

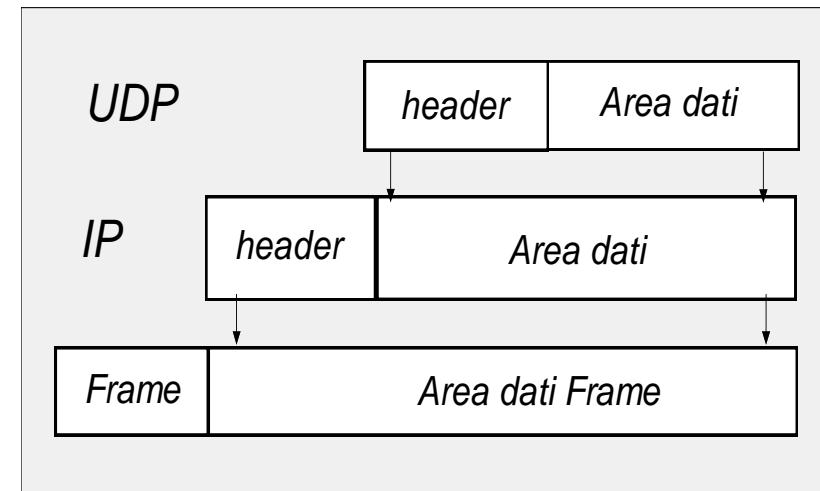
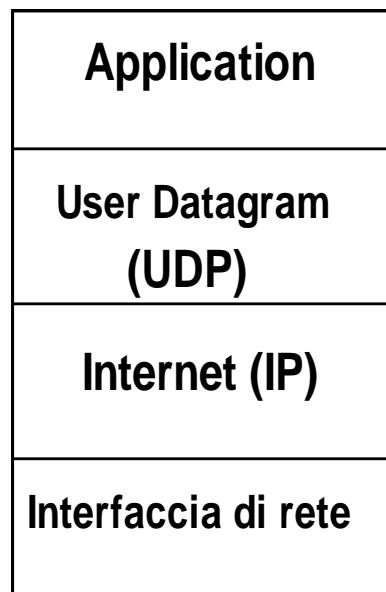
UDP deve distinguere tra più processi in esecuzione su un dato nodo connesso alla rete, processi identificati con numero di porta

UDP si appoggia a IP per consegnare i datagrammi

**indirizzo: indirizzo IP + numero di porta (16 bit)**

**NOMI: IP + Porte**

**LIVELLI CONCETTUALI**

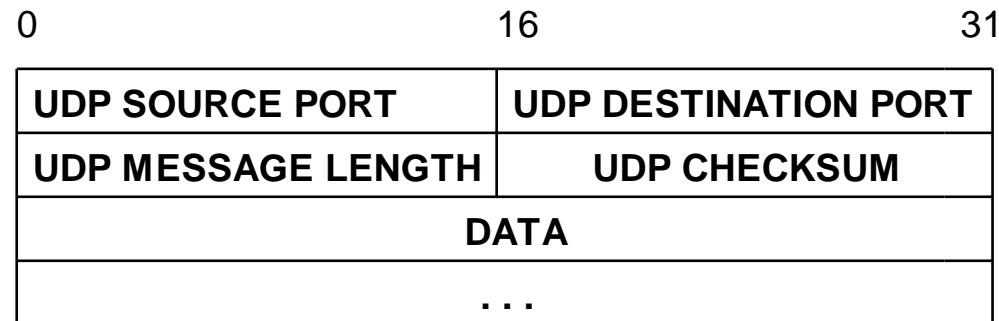


# USER DATAGRAM PROTOCOL

---

## UDP fornisce servizio unreliable e connectionless

- datagrammi possono essere persi, duplicati, pesantemente ritardati o consegnati fuori ordine
- il programma applicativo che usa UDP deve trattare i problemi



## Formato del datagramma UDP

I messaggi UDP sono composti di **header e area dati**, con **header composto solo di porte e lunghezza messaggio e checksum**

- Uno user datagram (UDP) è del tutto contenuto nell'area dati del datagramma IP, senza nessuna frammentazione
- Si noti la estrema limitazione in banda del protocollo e il limitatissimo overhead

# USER DATAGRAM PROTOCOL: PORTE

## Uso di porte

ogni programma ha una porta (o più porte) per inviare/ricevere datagrammi

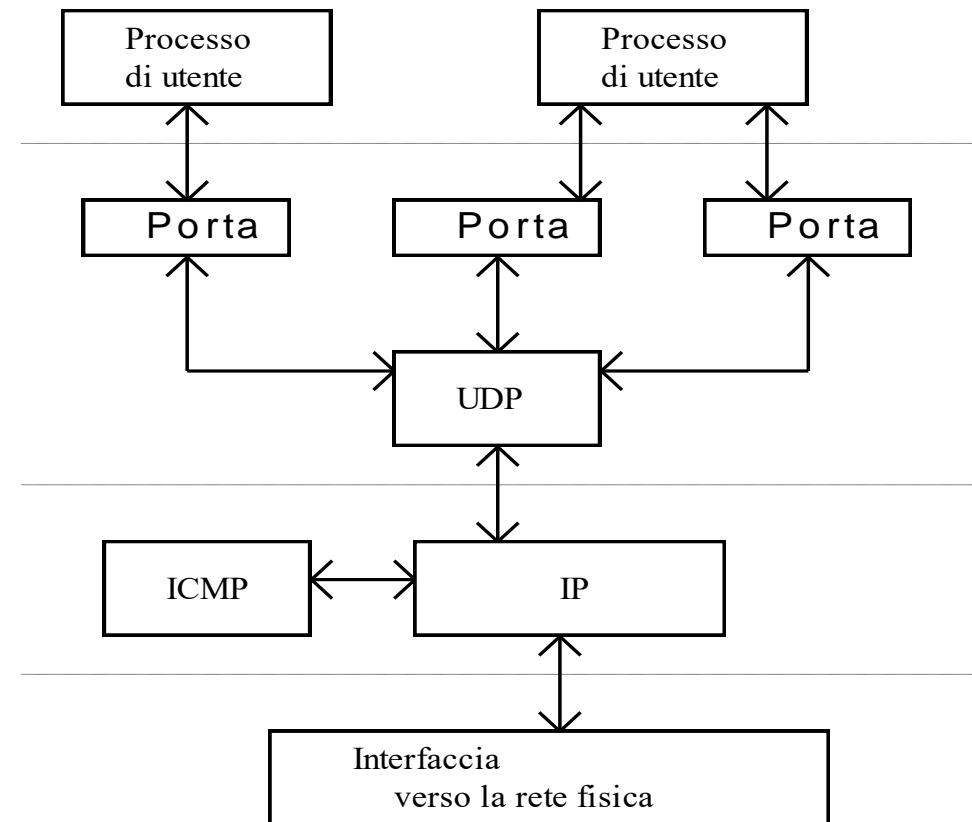
Decisioni di multiplexing/  
demultiplexing sulle porte  
unica driver UDP

### multiplexing

messaggi da più processi applicativi  
paralleli con un solo servizio IP

### demultiplexing

lo stesso messaggio recapitato alla  
porta corretta



# UDP: NUMERI PORTE

**NOMI STATICI** ⇒ Autorità Centrale pre-assegna i numeri di porta universalmente validi

**NOMI DINAMICI** ⇒ Assegnamento su necessità con binding dinamico: i numeri di porta non a priori, ma dati su richiesta

UDP/TCP adottano soluzione ibrida

- alcuni numeri di porta a priori (well-known port)
- gli altri assegnati dinamicamente



Alcune porte well-known  
e sempre rispettate per  
servizi standard

<b>0</b>	<i>Riservato</i>
<b>7</b>	echo
<b>9</b>	discard
<b>11</b>	users
<b>13</b>	daytime
<b>37</b>	time
<b>69</b>	tftp (trivial file transfer protocol)
<b>111</b>	Sun RPC protocol
<b>513</b>	who (demone di rwho)
<b>514</b>	system log

# TCP: ENTITÀ E CONNESSIONI

---

## Principali servizi TCP con porte distinte dalle porte UDP

TCP **permette la connessione end-to-end tra più processi di nodi distinti**, creando l'astrazione connessione come coppia di endpoint

- Un endpoint è definito come coppia di interi {host, port} con host l'indirizzo IP dell'host, e port della porta TCP

## La connessione è la quadrupla {host1, port1, host2, port2}

- i numeri di porta non sono esclusivi così come i nodi
- un numero di porta può essere condiviso da più connessioni
- un nodo può ospitare molte connessioni anche dalla stessa porta, se sono distinti gli altri elementi della quadrupla

## Sono connessioni distinte

- connessione {host1, port1, host2, **port2**}
- connessione {host1, port1, host2, **port3**}



Può esistere una sola connessione per quadrupla

# TCP: NUMERI PORTE

---

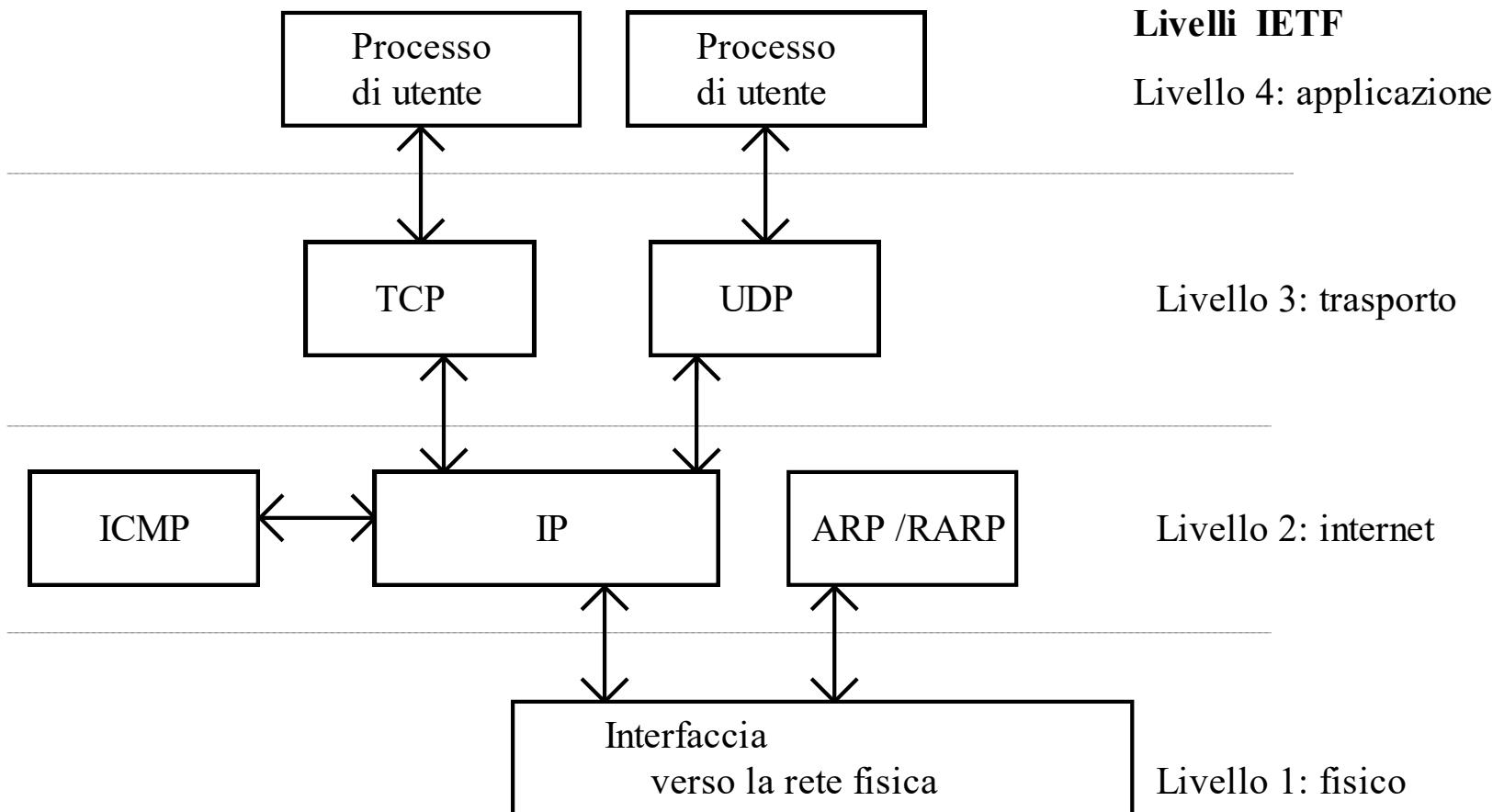
**Le porte TCP sono assegnate in modo ibrido**

Il quadro completo delle associazioni tra porte/servizi in  
**/etc/services**

PORT	PROTOC.	DESCRIZIONE
20	FTP-DATA	File Transfer Protocol (dati)
21	FTP	File Transfer Protocol
23	TELNET	Terminale remoto
25	SMTP	Protocollo di posta elettronica
80	HTTP	Protocollo web
119	NNTP	Protocollo news

# TCP: NUMERI PORTE

La driver TCP fa multiplexing/demultiplexing sulle porte



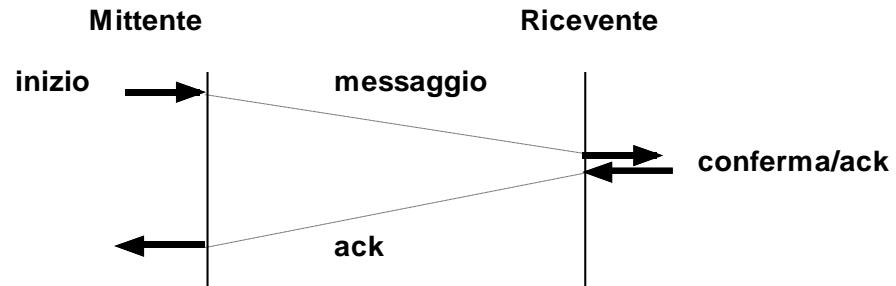
# COMUNICAZIONE: ARQ

Per la comunicazione e QoS, dobbiamo considerare due dimensioni, per svincolare il servizio dal ricevente e consentire asincronismi

- Affidabilità** ⇒ aspettare che il ricevente abbia ricevuto  
**Asincronismo** ⇒ non aspettare troppo

Automatic Repeat reQuest (ARQ)

**stop and wait**  
con ack del messaggio



**Notiamo che dobbiamo considerare anche casi di ritrasmissione con time-out, in caso di insuccesso**

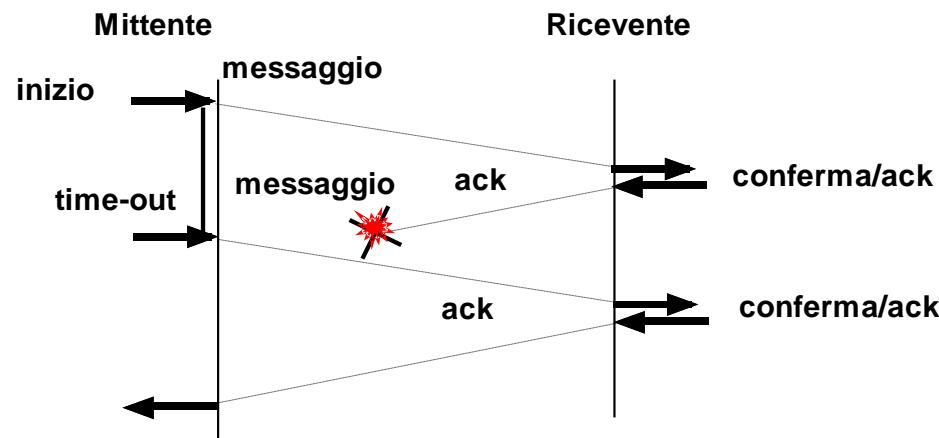
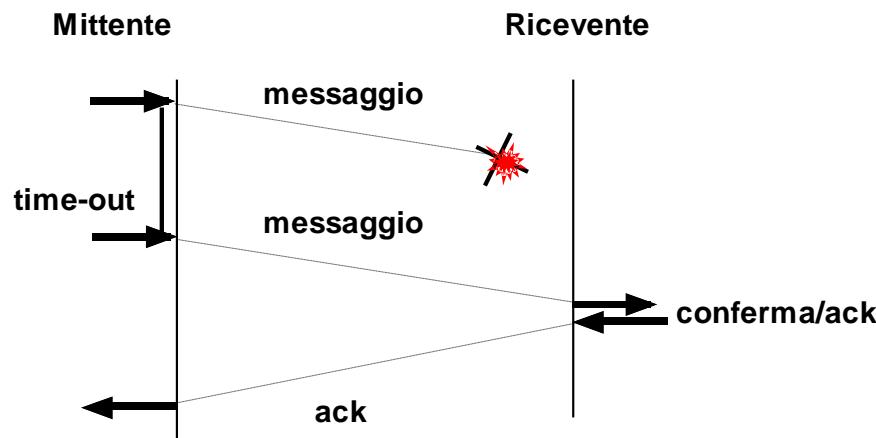
**Ancora di più abbiamo un estremo rallentamento del mittente, in caso debbano inviare molti messaggi allo stesso ricevente**



# COMUNICAZIONE: OLTRE ARQ

Naturalmente dobbiamo considerare la possibilità di **avere errori**:

- sia **perdite di messaggi**
- sia **conferme numerate per evitare messaggi compromessi**



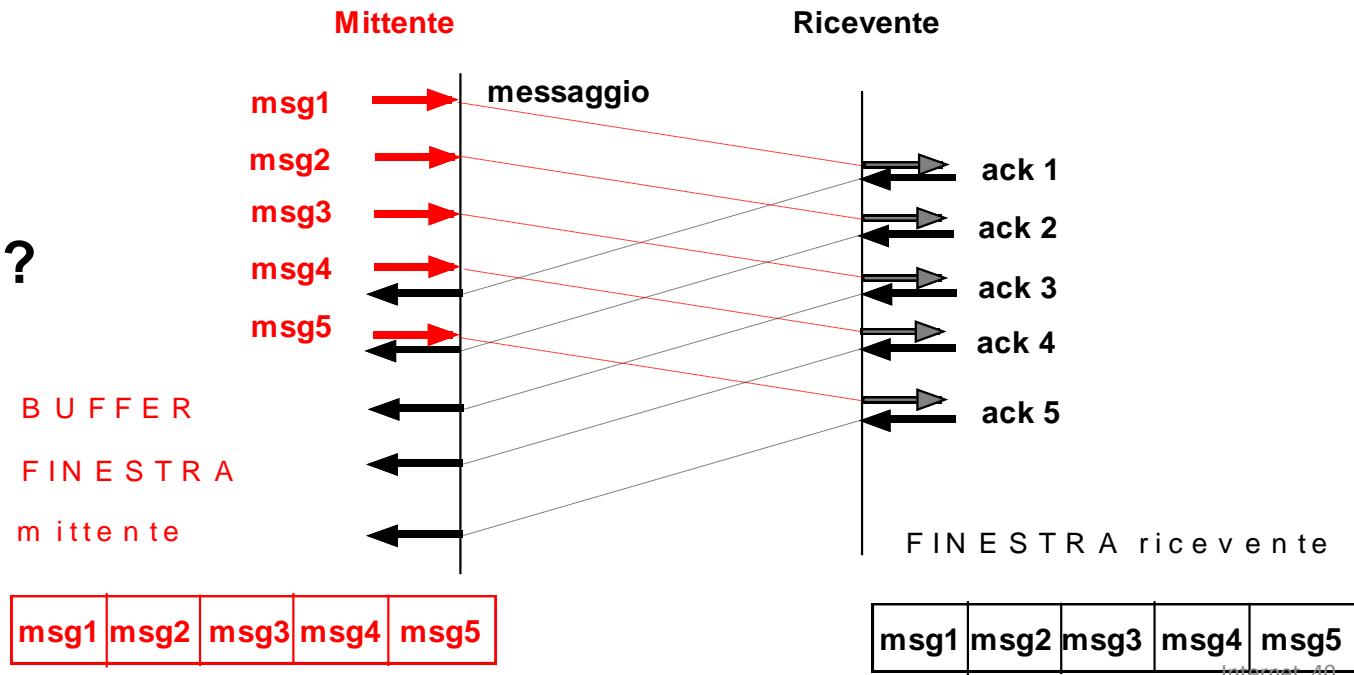
**Obiettivo dei protocolli è ottenere il massimo della qualità, cioè asincronicità e della autonomia di decisione tra i pari che devono comunicare**

# CONTINUOUS REQUESTS: OLTRE ARQ

Non si attende in modo sincrono della ricezione conferma (ack), ma si mandano messaggi in modo ripetuto (**Continuous Requests**)

- Il mittente manda messaggi che sono mantenuti fino a saturare la **memoria disponibile** (finestra buffer) e sono scartati solo alla conferma
- Il mittente **scorre la finestra** in caso di conferma (all'acknowledgement)
- Attesa del mittente solo a **finestra piena**
- Il ricevente passa i messaggi all'applicazione solo in **ordine giusto**

La dimensione  
della finestra  
imposta da chi?



# CONTINUOUS REQUESTS

---

Protocollo più complesso rispetto al caso ARQ

**Le conferme da sole (messaggio cortissimo) sono overhead**

- per full-duplex, gli ack sono mandati in **piggybacking sul traffico opposto**

E in caso di errore o di un messaggio non arrivato, mentre altri messaggi successivi arrivati?

## SELECTIVE RETRANSMISSION

- attesa dell'esito dei messaggi tenendo conto degli ack ricevuti e anche ack negativi (dovuti al time-out del ricevente) e ritrasmissione di quelli persi

## GO-BACK-N

- attesa di ack e ritrasmissione (solo con time-out al mittente) e tenendo conto di ack del ricevente (che salta i non ricevuti); il mittente scarta i messaggi successivi non in sequenza e li rimanda tutti al ricevente

go-back **confonde messaggi non in ordine con perdite**

**(TCP usa go-back-N ottimizzato e ack cumulativi)**

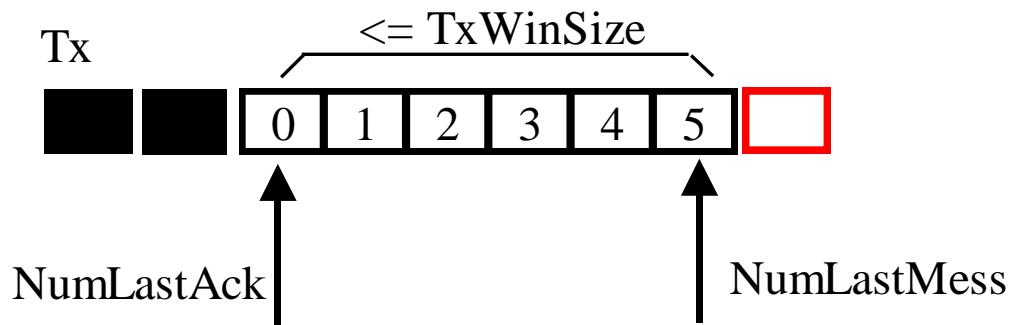
# SLIDING WINDOW: TX

Nei protocolli **continuous requests**, ogni direzione di trasmissione usa una finestra scorrevole (**sliding window**) per la gestione della memoria di bufferizzazione

il Mittente (TX) numera ogni messaggio con **NumSeq**  
accetta una dimensione della finestra **TXWinSize**  
mantiene il valore dell'ultimo messaggio inviato **NumLastMess**  
mantiene il valore dell'ultimo ack ricevuto **NumLastAck**

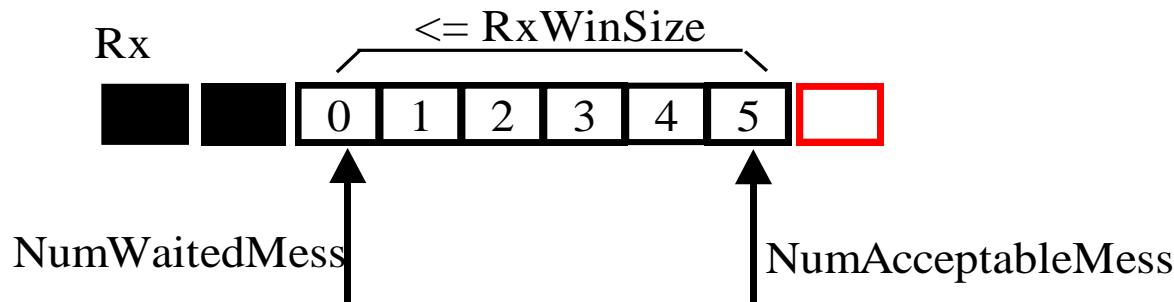
obiettivo del mittente è mantenere

$$\text{NumLastMess} - \text{NumLastAck} + 1 \leq \text{TXWinSize}$$



# SLIDING WINDOW: RX

- il Ricevente (RX) decide e mantiene una dimensione della finestra **RXWinSize**
- il numero del prossimo messaggio atteso **NumWaitedMess**
- il numero dell'ultimo messaggio confermabile **NumAcceptableMess**



Obiettivo del ricevente è mantenere

$$\text{NumAcceptableMess} - \text{NumWaitedMess} + 1 \leq \text{RXWinSize}$$

Nei protocolli continuous requests, molte politiche e decisioni diverse:  
**la decisione della dimensione del buffer è sempre del ricevente che deve allocarla e mantenerla per i messaggi**

# TCP: TRANSMISSION CONTROL PROTOCOL

---

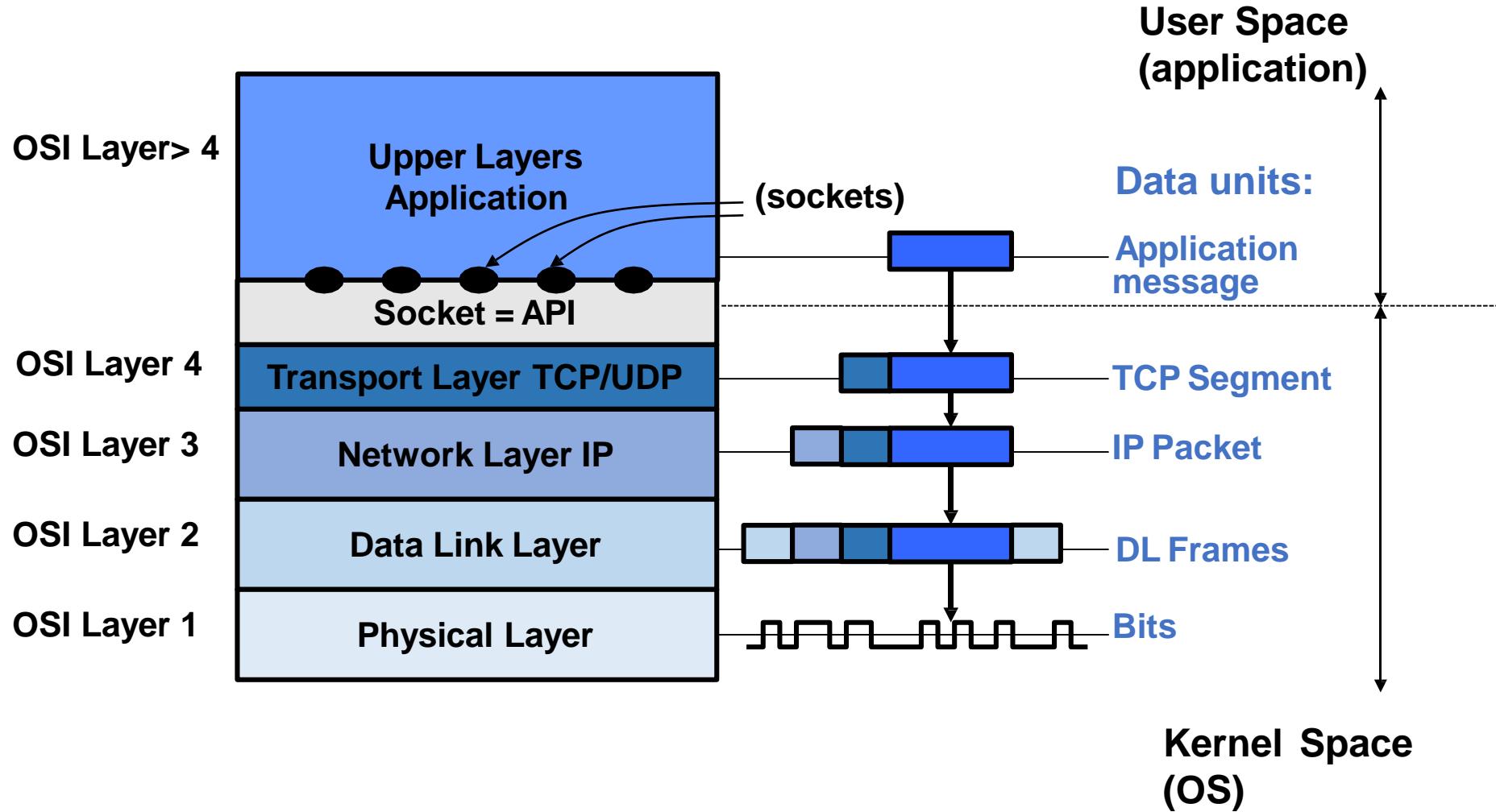
TCP fornisce un servizio di trasmissione dati affidabile basato su

- **reliable stream full duplex**
- **connessione o canale virtuale bidirezionale**
  - la **connessione end-to-end** garantisce che il messaggio passi dalla memoria del mittente al destinatario con successo e che si mantenga il flusso
  - Lo **stream**, ossia il flusso è costituito di **dati in ordine preciso e non alterabile**, anche se sono riconosciuti due tipi di dato che seguono lo stesso ordine di stream
    - flusso di dati non strutturato (**byte stream**) e dati **normali**
    - dati **prioritari** in banda limitata (**1 solo byte**)
  - **banda disponibile per i dati normali, banda limitata per i dati urgenti** con avviso appena **dati urgenti sono presenti sul flusso**



**La connessione TCP NON impegna i nodi intermedi  
si usano solo le risorse dei nodi degli end-user**

# PROTOCOLLO TCP - LIVELLI



# PROTOCOLLO TCP

---

Il **protocollo TCP** si basa su alcuni principi e vincoli da rispettare:

- **formato dei dati trasmessi** (segmenti con header fissato)
- possibilità di **dati urgenti**
- regole per la **bufferizzazione e l'invio degli acknowledgement** (sliding window) e relativo formato
- possibilità di **comporre messaggi e decomporre in segmenti**
- meccanismi di **de/multiplexing** (vedi UDP) attraverso il concetto di porta per distinguere più processi su uno stesso host

**La realizzazione si basa sulla implementazione della connessione e sulla comunicazione, permettendo servizi che devono occuparsi di**

- stabilire la connessione
- scambiare dati sulla connessione
- chiudere la connessione

# FORMATO HEADER TCP

Nell'header TCP, 5 parole da 4 byte (almeno)

- **1 parola** per le porte
- **2 parole** per i tag dei dati inviati e da ricevere
- **1 parola** code bit window
- **lunghezza header**
- **1 parola** puntatore urgente
- **checksum controllo**
- Opzioni eventuali

0	4	10	16	24	31		
SOURCE PORT			DESTINATION PORT				
SEQUENCE NUMBER							
ACKNOWLEDGEMENT NUMBER							
HLEN	RSRVD	CODE BIT	WINDOW				
CHECKSUM			URGENT POINTER				
OPTIONS (IF ANY)				PADDING			
DATA							
...							

# FORMATO TCP: CODE BIT

L'header del segmento TCP è costituito da 20 byte (5 parole)

## CODE BIT

### URG

dato urgente nel flusso

0      4      10      16      24      31

### ACK

ack nel segmento

### PUSH

invio immediato segmento

### RST

reset di una connessione

### SYN

si stabilisce la connessione

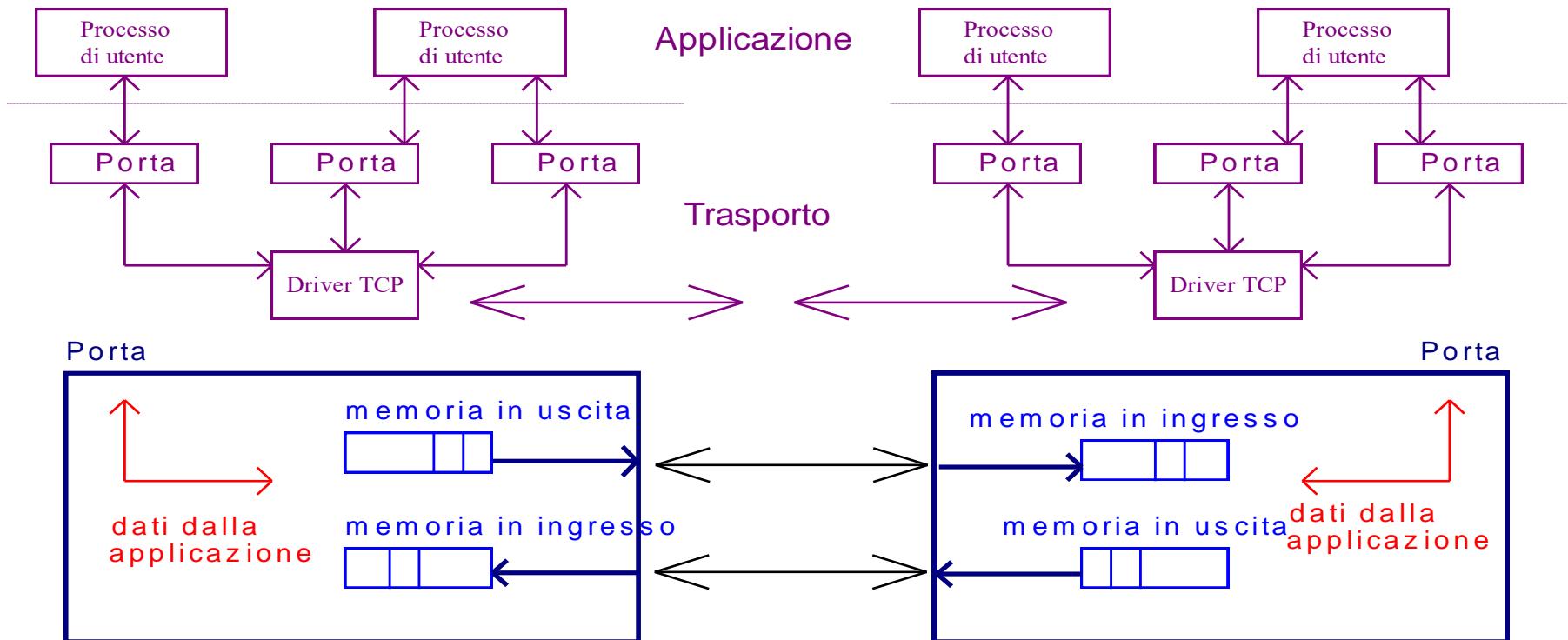
### FIN

termine della connessione

SOURCE PORT	DESTINATION PORT		
SEQUENCE NUMBER			
ACKNOWLEDGEMENT NUMBER			
HLEN	RSRVD	CODE BIT	WINDOW
CHECKSUM		URGENT POINTER	
OPTIONS (IF ANY)		PADDING	
DATA		...	

# ENTITÀ

Nelle connessioni TCP mettiamo in gioco i diversi livelli



Per ogni porta connessa punto a punto si alloca memoria di supporto

Ogni connessione permette di scambiare un flusso full-duplex di dati normali e un (limitato) flusso full-duplex di dati urgenti

# CODE BIT

---

**L'header del segmento TCP riporta i code bit o flag**

- ACK** se il bit è ad uno, allora il segmento riporta un ack significativo nel campo relative
- URG** se il bit è ad uno, allora nel flusso mittente c'è un bit urgente e questo viene segnalato da ogni header di segmento inviato al ricevente (il campo urgent pointer dice la distanza dalla posizione corrente nel flusso)
- PUSH** se il bit è ad uno, allora il mittente segnala di processare e di passare al livello applicativo tutti i dati (anche buffer vuoto al mittente)
- RST** se il bit è ad uno, allora ci sono dei problemi nella connessione e il ricevente deve tentare misure estreme come resettare (tipicamente notificando all'applicazione)
- SYN** se il bit è ad uno, allora si sta stabilendo la connessione (apertura iniziale a tre fasi)
- FIN** se il bit è ad uno, allora si sta iniziando la chiusura da parte di uno dei due endpoint, del mittente del bit FIN (chiusura a quattro fasi)

# TCP: COMUNICAZIONE

---

**TCP può spezzare i messaggi applicativi in segmenti di dimensione variabile, e tende a frammentare messaggi in segmenti**

- né troppo **corti**: grosso overhead di trasmissione
- né troppo **lunghi**: frammentazione a livello di IP e possibili perdite

**TCP usa **CONTINUOUS REQUEST** per efficienza e affidabilità**

- I messaggi prevedono ack, che essendoci traffico nei due sensi, gli ack sono inseriti sul traffico in direzione opposta (piggybacking)

**USO di finestra scorrevole, espressa in byte, determinata e decisa dal ricevente e comunicata per ogni invio di segmento**

- il **mittente invia segmenti fino a saturare la finestra** senza conferma di ricezione, poi si deve fermare, il destinatario invia ack alla ricezione
- se i segmenti sono confermati, la finestra scorre e si avanti nel flusso
- se scade il time-out di un segmento, si reinvia
- gli ack potrebbero arrivare non in accordo all'ordine di trasmissione

# TCP: RITRASMISSIONE

---

**TCP usa GO BACK-N, in caso di non ricezione di un segmento**

- il ricevente può scartare quelli successivi e attendere il segmento mancante
- il mittente deve rimandare i segmenti da quello che manca
- reinvio anche ripetuto fino ad una eccezione (fallimento)
- il ricevente deve favorire il rinvio di segmenti mancati

**In realtà il ricevente ottimizza e non scarta immediatamente i segmenti fuori ordine (ma li mantiene se può per integrarli)**

Parametri decisi dal protocollo e non visibili

- DOPO quanto tempo si ritrasmette
- QUANTE VOLTE si esegue le ritrasmissione
- COME si frammentano i segmenti

**Il protocollo a stream può rimandare parti del flusso ossia segmenti con dimensioni diverse senza garanzie di lunghezze predefinite o stabili**

# TCP: CONFERME

## TCP conferma con ack cumulativi

- Arrivo di ack di un messaggio implica che sono arrivati anche i precedenti
- Perdita di ack non forza ritrasmissione

### Svantaggio

- un ack cumulativo dice poco sullo stato del ricevente
- al mittente, ack indica sempre la stessa posizione nello stream ricevente (ultimo byte arrivato in ordine, anche se successivi fossero arrivati)
- con modalità ack selettivo si potrebbe aspettare l'ack dopo la trasmissione e reinviare solo quelli mancanti



In questo caso si deve reinviare tutto, anche quelli già ricevuti dal ricevente (**GO-BACK-N**)

**Si tende a rimandare solo il primo, poi si aspetta per verificare che il ricevente non possa mandare ack cumulativo di una parte di flusso successiva**

# TCP: RIASSUNTO INIZIALE...

---

TCP, rispetto ad altri protocolli è a streaming di byte, e:

- **lavora con finestra di dimensione variabile specificata del ricevente per ogni verso**
- usa **byte** per dimensione della **finestra**
- intende gli **ack in modo cumulativo**
- un ack specificato del ricevente porta l'indicazione di tutto ciò che è stato ricevuto nello stream fino al momento dell'ack
- **in caso di perdita**, si continua a mandare ack per l'ultimo ricevuto
- **ritarda i messaggi** che vengono raggruppati in un segmento locale prima dell'invio (anche gli ack)
- tende a **non mandare messaggi corti** raggruppandoli al mittente peggioramento del tempo di risposta specie in caso di interattività definizione di un time-out oltre il quale il messaggio corto viene inviato
- usa **piggybacking per gli ack**
- gli ack sono ritardati in attesa di traffico in verso opposto

# TCP: FASI DI OPERATIVITÀ

---

## fase iniziale - three-way handshaking

- in cui si stabiliscono una serie di parametri operativi per la connessione e si prepara l'avvio

## fase di comunicazione – transitorio e regime

**transitorio iniziale** si comincia a lavorare

- ... senza essere subito a regime in fase iniziale esplorativa
- **regime** in varie condizioni operative diverse
  - si devono considerare situazioni di congestione individuando o prevenendo i colli di bottiglia fino a ristabilire una situazione normale o fino ad un abort della connessione

## fase finale – chiusura mono e bidirezionale

- chiusura manifestata da uno dei due pari e accettata dall'altro
- operatività con canale monodirezionale di dati, ma con messaggi di controllo in entrambe le direzioni

# TCP: FASE INIZIALE

---

Per stabilire la connessione TCP il mittente attua un protocollo per realizzare la connessione tra le due driver di protocollo dei due nodi **three-way handshake**

- tre fasi di comunicazione per il coordinamento iniziale tra mittente A che gioca un ruolo attivo e ricevente passivo B con **valori random** per etichettare il byte d'inizio del flusso di comunicazione nei due versi **ogni byte è collocato nel suo stream**

## PRIMA FASE

A invia a B il segmento con SYN e richiede la connessione (SYN nell'header del segmento e X valore iniziale del flusso scelto da A)

## SECONDA FASE

B riceve il segmento SYN e ne invia uno identico ad A con ACK (anche del valore mandato da A) anche SYN con Y valore scelto da B per il suo verso

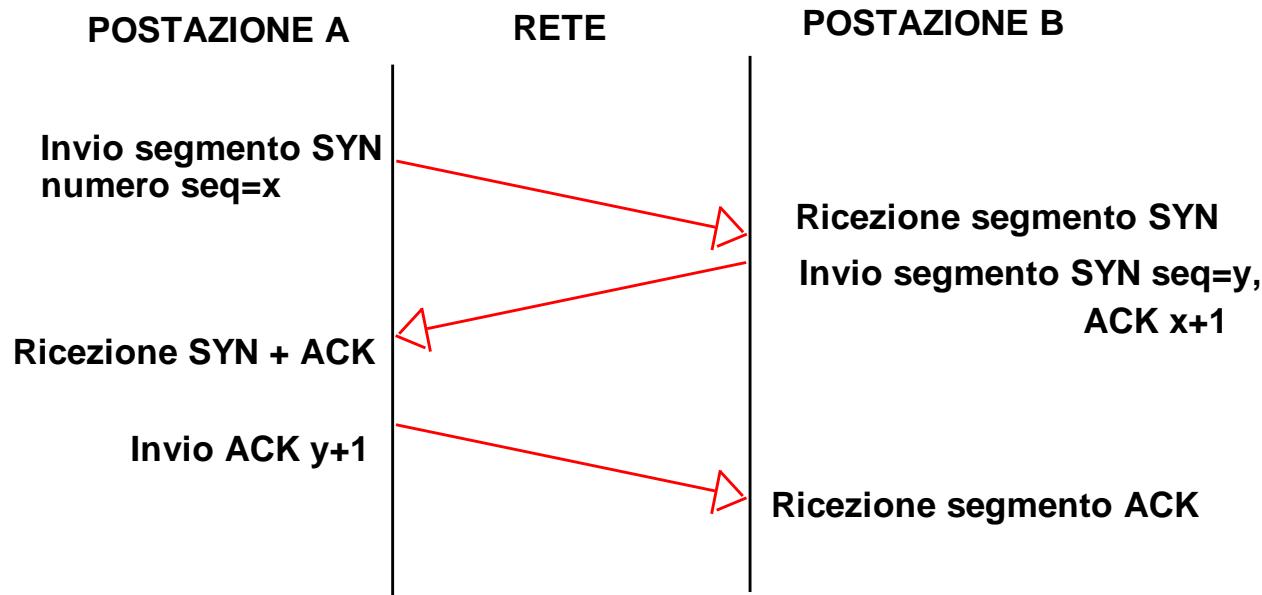
## TERZA FASE

A riceve il segmento SYN ed ACK e conferma la ricezione a B attraverso un ACK a sua volta

# TCP: THREE-WAY HANDSHAKE

Per ottenere la semantica at-most-once sono necessarie le tre fasi di coordinamento

Ogni nodo invia un messaggio ed ha conferma del valore inviato



Messaggi di gestione (syn e ack in rosso) senza dati

Invio anche di window e opzioni

Perché non ci si accontenta di due fasi come nel C/S tipico?

# TCP: PERCHÉ THREE-WAY

---

**Si usano tre fasi di coordinamento**

In linea di principio, ogni messaggio di un protocollo richiede una conferma

**Così i protocolli dovrebbero prevedere infinite fasi**

Dobbiamo capire che protocolli sono in modo comune sono implementati in modo ingegneristico, pensando a **time out e ripetizione del messaggio**

**Si faccia riferimento al normale C/S**

**Il cliente rimanda il messaggio dopo un timeout**

Le tre fasi sono motivate per la necessità specifica di **misurare il tempo andata e ritorno tra i due pari da ciascuno** in modo che

**Il C misura il proprio tra le prime due fasi**

**Il S misura il proprio tra secondo e terza fase**

# THREE-WAY HANDSHAKE: ACCORDO

---

**NEGOZIAZIONE** a tre fasi per stabilire proprietà se entrambi i nodi disponibili alla connessione

- **BIDDING** (offerta senza rifiuto)

Ogni pari decide in modo unilaterale il proprio verso della connessione; il pari deve accettare (e chiudere subito dopo)

**Coordinamento sulla sequenza iniziale di valori:**

- numeri di **porta disponibili**

- numeri di **inizio per i flussi (X e Y) scelti in modo casuale**

scelta casuale di un numero da cui iniziare la numerazione, comunicato all'altra per ogni direzione di flusso: si vuole evitare di utilizzare vecchi segmenti di vecchie connessioni omologhe!!

- **tempo di trasmissione e risposta** (time-out)

Ognuno manda e riceve una risposta per il calcolo del proprio timeout

- **finestra di ricezione** (window), ...

# FASE INIZIALE - ANCORA

---

**E se si perde un messaggio nelle prime fasi? Si rinvia ...  
con che tempi di timeout?**

**Si attua un time-out con intervalli crescenti**

- normalmente il primo dopo 5,8 sec, poi 24 sec., ecc  
oltre, si chiude

**In fase iniziale si possono negoziare altre opzioni:**

- accordo sul **segmento medio** o MSS (Maximum Segment Size) dimensione del blocco di dati massimo da inviare
- default 536: se maggiore, migliori performance
- **fattore di scala della finestra**
- richiesta di **tempo e risposta** per il coordinamento degli orologi

Sono possibili anche **azioni simultanee di apertura** da parte di due entità che dovrebbero portare alla stabilire una sola connessione senza corse critiche

# TCP: FASE FINALE

---

**CHIUSURA - chiusura a 4 vie - 4-way handshaking  
(sempre solamente 4 fasi in chiusura)**

Si prevede una semplice **operazione di chiusura graceful**

**Chiusura monodirezionale di output** ossia definitiva per un solo verso (il verso di autorità) senza perdita dei messaggi in trasferimento e di quelli in arrivo ... attuata per ogni verso

**Se ad esempio A chiude nel suo verso di uscita:**

A comunica a TCP di non avere ulteriori dati e chiude  
TCP chiude comunicazione solo nel verso da A a B

**I dati che precedono la fine sono ricevuti prima della fine  
dello stream da A a B**

**TCP permette il passaggio di ack su canale intenzionalmente  
chiuso**

La parte di controllo è aperta da A a B (flusso di ack)

**TCP permette la comunicazione in verso opposto**

Se B non ha terminato, i dati continuano solo da B ad A

# TCP: CHIUSURA IN 4 FASI

A invia segmento FIN in ordine dopo l'invio dei dati precedenti

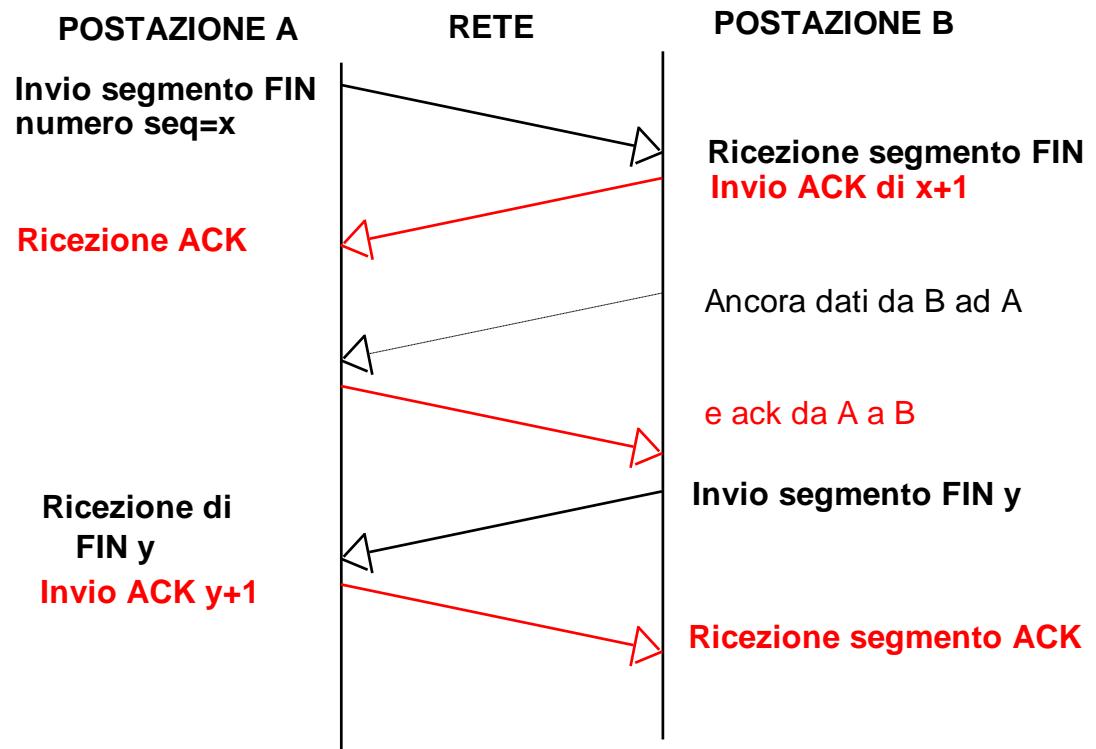
TCP aspetta a dare corso alla chiusura, ma invia da A a B solo ack

Si manda traffico applicativo da B ad A

Al termine del traffico applicativo da B ad A

B invia ad A il segmento FIN che informa della disponibilità a chiudere la connessione

L'ultimo passo conferma da A a B della ricezione del segmento FIN e la chiusura totale della connessione



# SHUTDOWN OUTPUT vs. CHIUSURA

Si noti che il protocollo di chiusura come coppia di due fasi, da A le prime due, e da B nella seconda parte

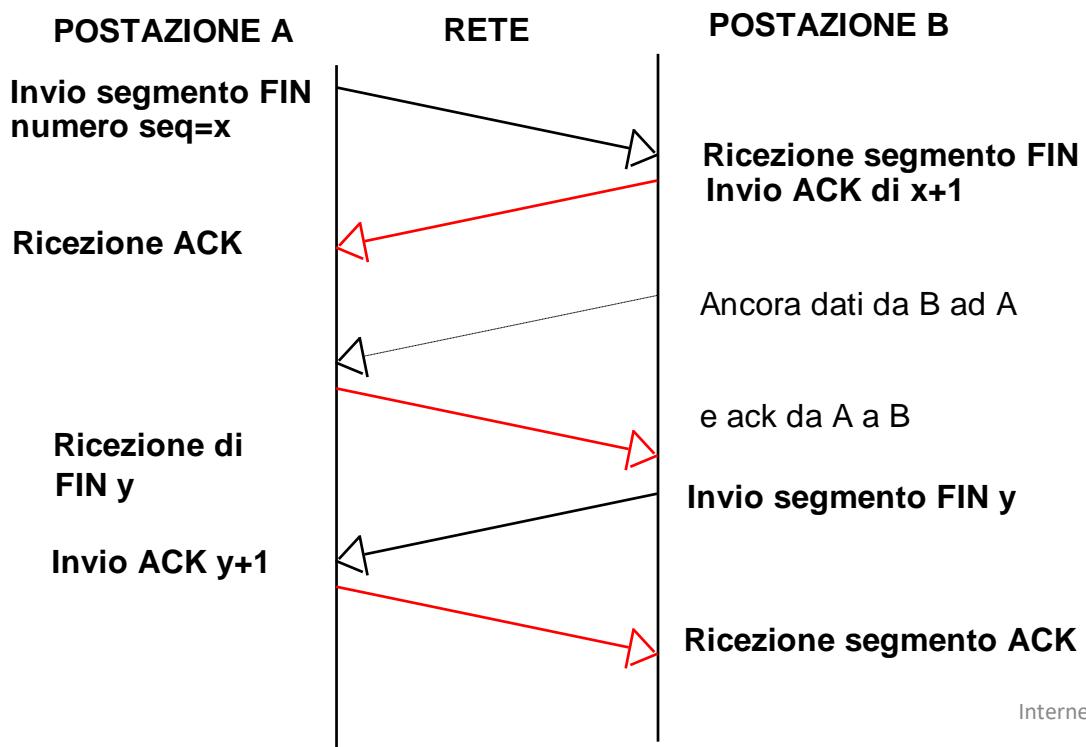
Ogni pari attua la sua chiusura secondo la propria decisione (con una shutdown in output) e determina il proprio verso attivo di connessione

I dati in uscita terminano tutti in modo controllato e sono tutti consumati

Non ci sono dati in ingresso non processati dal pari corrispondente

In caso di **close** invece,  
chi chiude attua  
sia una chiusura in out  
sia una chiusura in in

rendendo inutili tutte le  
trasmissioni dei dati che  
non accetta più in input



# AUTOMA A STATI PER TCP

---

## Macchina a stati

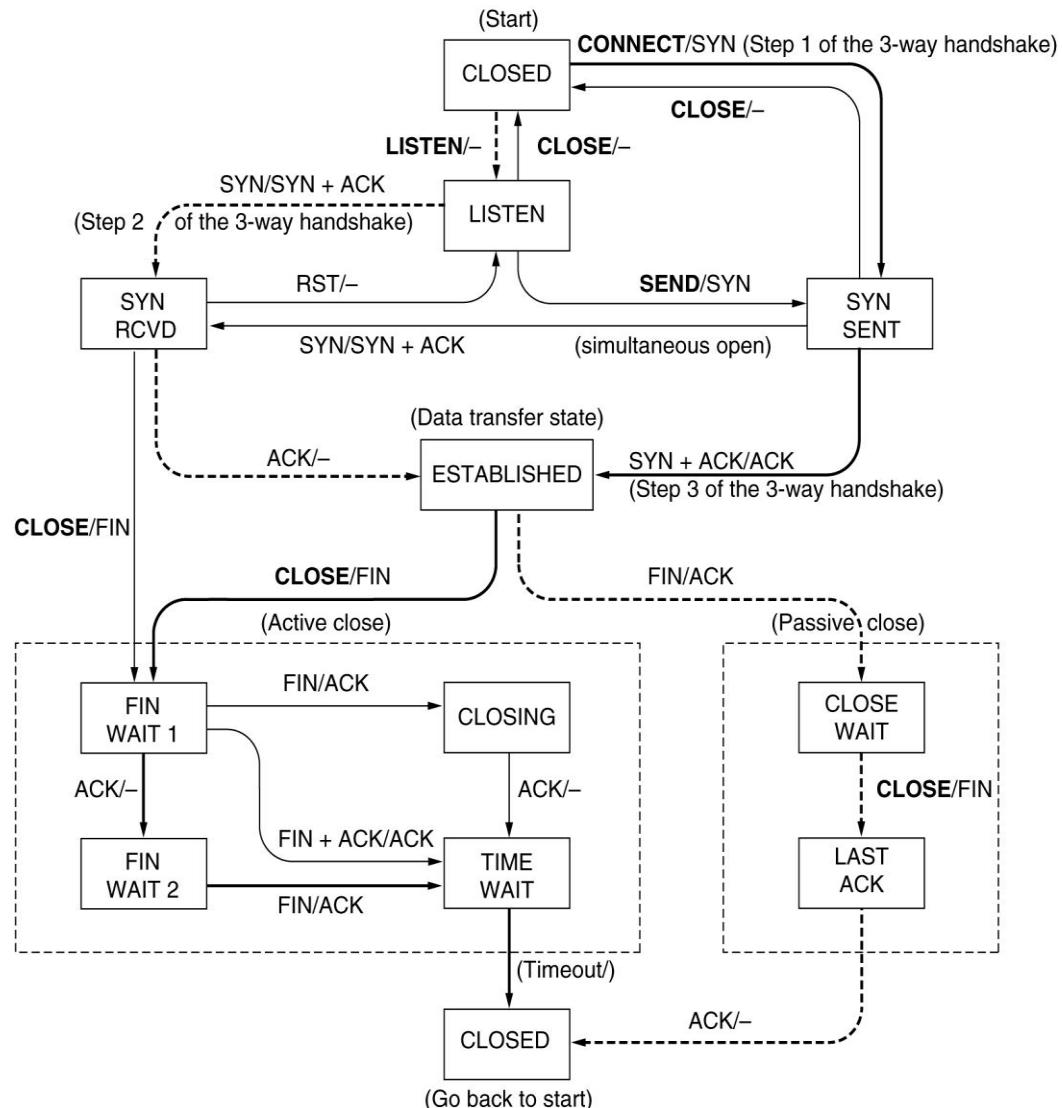
I protocolli **possono essere descritti come automi a stati** per mappare le diverse fasi di operatività della connessione

State	Description
CLOSED	No connection is active or pending
LISTEN	The server is waiting for an incoming call
SYN RCVD	A connection request has arrived; wait for ack
SYN SENT	The client has started to open a connection
ESTABLISHED	Normal data transfer state
FIN WAIT 1	Client has said it is finished
FIN WAIT 2	Server has agreed to release
TIMED WAIT	Wait for pending packets (“2MSL wait state”)
CLOSING	Both sides have tried to close simultaneously
CLOSE WAIT	Server has initiated a release
LAST ACK	Wait for pending packets

# STATI IN TCP

Macchina a stati  
per mappare le  
operazioni  
sulla connessione

In particolare per le fasi  
di  
inizio della connessione  
(inizio da listen a  
established)  
e di fine della  
connessione  
(fine da established  
a closed)



# TCP: GESTIONE FASI ANOMALE

**Sono considerati eventi anomali possibili i casi di fallimento ripetuto**

La gestione può avvenire attraverso un segmento di **reset** inviato per rilevare una situazione anomala ad esempio

- richiesta di connessione senza server
- un reset della connessione stabilita per abortire la stessa (con perdita dei dati)

Tipicamente dopo alcuni invii di tentativo a vuoto, si attua un reset per tentare un ripristino o chiudere tutto



**NOTIAMO ANCORA che:**

- La connessione esiste solo negli endpoint
- solo in caso di guasto e di azioni ripetute di recovery, si stabilisce di chiudere in modo abortivo e unilaterale

# TCP: INIZIALE

---

La connessione è una entità diversa dai singoli endpoint e anche se ci sono solo le driver degli estremi a realizzarla, **si deve tenere in conto la situazione intermedia per non causare eccessi o problemi di congestione**

La operatività sulla connessione avviene solo dopo avere cominciato in modo graceful e tentando di andare verso la situazione di regime nel modo più graduale possibile, ad esempio ...

- Anche se **sulla connessione il cliente (o il servitore) volesse** mandare subito una ingente quantità di dati, questa viene ritardata per evitare di avere comunicazioni iniziali pesanti e per non produrre situazioni pericolose di congestione iniziale attraverso procedure graduali
- **Ad ogni grossa variazione di operatività**, il passaggio da una fase ad un'altra viene gestito secondo lo stesso principio della minima intrusione tentando così di evitare peggioramenti globali

# TCP: REGIME

---

**A regime si fanno continui aggiornamento dei valori in base alla situazione corrente rilevata**

## Calcolo del time-out principale o Round-Trip Time

- Dopo il calcolo iniziale da parte di ognuno dei due
- Si ricalcola per ogni segmento, in base al tempo di percorrenza medio RTT andata e ritorno

Si sono diffusi diversi algoritmi di calcolo del time-out sempre tenendo conto di criteri di minima intrusione e dell'efficienza

Ricalcolo del Time-out in base a una formula del tipo (**Karn**)

$$\text{Timeout} = \alpha * \text{Intervallo precedente} + \beta * \text{Intervallo corrente}$$

**il timeout tiene conto della storia pregressa e del valore ricavato correntemente con pesi diversi, per non essere né troppo reattivo né troppo conservativo**

# TCP: TIMEOUT PRINCIPALE

---

**Il timeout principale viene continuamente ricalcolato**

il time-out principale viene calcolato multiplo di 100, 200 o 500 ms  
in modo da dovere gestire una granularità limitata

**Problemi in caso di ricalcolo su ritrasmissioni**

- In caso di **invii ripetuti di segmento**, un ack in arrivo è per il primo o per un messaggio ritrasmesso dal punto di vista del calcolo?
  - In caso di **associazione al primo**, potremmo cambiare il timeout in modo troppo reattivo per una semplice perdita di un messaggio
  - In caso di **associazione al secondo**, potremmo non cambiare il timeout non adeguandoci alla nuova situazione di rete



**Nessun ricalcolo in caso di perdita potenziale di messaggio e re-invio da parte del mittente: il ricalcolo si fa solo in caso di successo senza ritrasmissione**

# TCP: ALTRI TIMEOUT

---

**Il timeout principale è la base di molti parametri temporizzati per la connessione**

**Distinguiamo mittente e ricevente**

- il **ricevente differisce i messaggi corti di ack** in modo da sfruttare il piggybacking sul traffico, usando un time-out per limitare il ritardo massimo: dopo tale timeout si invia un segmento ad-hoc di controllo
- il **mittente differisce i byte applicativi** e li mantiene fino ad avere raggiunto un segmento di dimensione media (MSS): dopo un certo timeout, si invia il messaggio corto in ogni caso per non incorrere in troppo ritardo
- **entrambi gli endpoint ritardano i messaggi alla applicazione fino ad avere segmenti medi**: in caso di PUSH si invia e riceve rapidamente
- **entrambi gli endpoint** in caso non ci sia traffico **mandano messaggi di verifica** del pari in modo da sapere la situazione corrente della connessione: dopo un intervallo (lungo) si invia un segmento di controllo

# TCP: FLOW CONTROL

---

**Il controllo di flusso è fondamentale in Internet in cui ci sono connessioni con macchine molto diverse fra loro**

Sono meccanismi fondamentali di coordinamento:

- **la finestra**

- La **dimensione della finestra** viene inviata per ogni segmento e comunica al pari quali siano le esigenze di memoria della connessione
- una finestra a 0 significa di non inviare alcune segmento
- Ogni pari comunica all'altro la propria situazione con la finestra

- **la dimensione preferenziale dei segmenti da inviare**

- attesa di dati prima di inviarli fino ad avere un segmento che sia conveniente inviare (**Maximum Segment Size o MSS** come opzione TCP)

**Si deve evitare di avere trasmissioni di messaggi corti**

**Silly window finestre limitate e messaggi brevi**

in genere non si fanno azioni sotto una soglia e non si mandano finestre troppo piccole, così come non si mandano segmenti troppo corti

# TCP: ALGORITMO DI NAGLE

---

**Una scelta praticata per evitare messaggi corti è stato l'algoritmo di Nagle**

si ammette di avere pendente senza ack al più un solo messaggio corto - retroazione automatica per non inviare messaggi corti in eccesso

Applicazioni come XWindow disabilitano l'algoritmo di Nagle per ottenere una migliore interattività come possibilità Utente

**La politica Nagle è spesso disabilitata ora**

Le applicazioni possono anche cercare di superare la trasparenza di TCP, usando **segmento con code bit PUSH**

il segmento inviato immediatamente e portato all'applicazione prima possibile

**segmento con indicazione di informazioni urgenti sul flusso (code bit URG)**

se ne segnala la posizione nel flusso, e il ricevente deve consumare i dati per arrivare quanto prima al byte di urgente

# TCP: DATI URGENTI

In caso di dati urgenti, che sono al massimo 1 byte, se ne segnala immediatamente la presenza e anche la posizione nel flusso

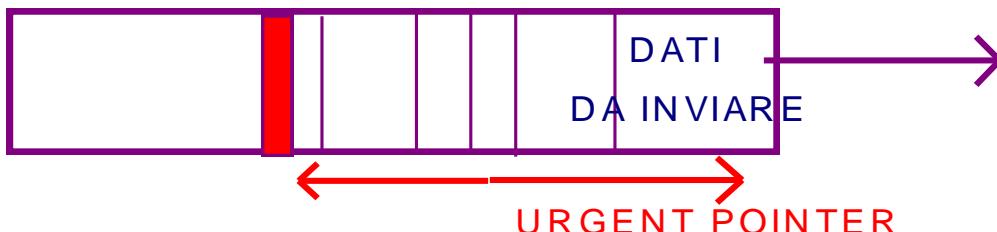
Per ogni flusso, si hanno due versi

## FLUSSO in USCITA

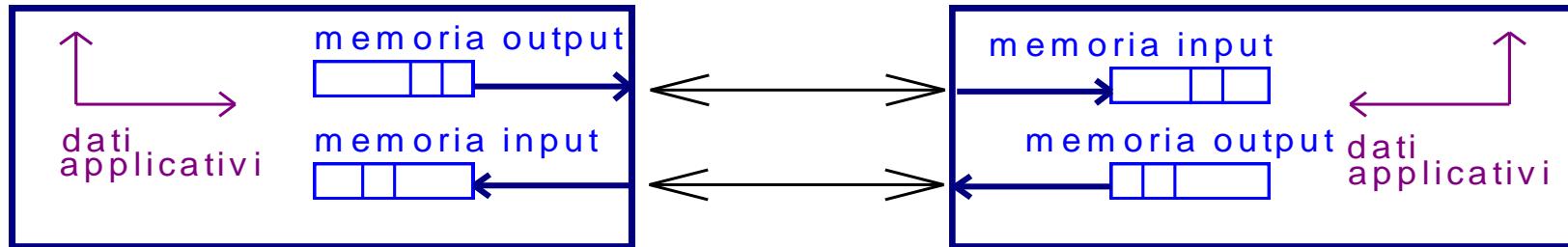
Per un verso, in uscita

# corrente

# posizione iniziale



## SEGMENTO DATI HEADER TCP



# TCP: REGIME NORMALE

---

A regime ogni segmento inviato produce coordinamento con il pari attraverso l'header del segmento stesso

Ogni **segmento** invia sempre informazioni di controllo al pari

- sia la propria **posizione nel flusso**
- sia la **posizione nel flusso ricevuto con ack**
- la **finestra di ricezione corrente** nella propria direzione

Il **ricevente** adegua i propri parametri come

- la **dimensione della sliding window** (di cui è mittente)
- i **time-out, misurati e riadeguati**

in situazione normale ... (non PUSH, non URG, ...)

**La connessione TCP non usa risorse se non si inviano messaggi**

- Timer per garantire l'operatività invio di un messaggio di keep-alive inviato ad intervalli molto distanti (7200 sec.)

# TCP: CONGESTIONE

---

**Il caso di congestione è critico per TCP**

**Congestione** come scenario in cui non si riescono più a consegnare dati in tempi utili (rispetto alla operatività corrente)

- Intendiamo con **congestione** una situazione che può essere sia dipendente dai soli endpoint della connessione stessa, sia da una più ampia situazione della intera rete
- Tutti i **router** sono a **buffer pieni** e nessuno scambio può più avvenire, fino alla de-congestione
- Abbiamo **forzato troppo il cammino della connessione** e abbiamo congestionato solo le nostre risorse locali

Nel secondo caso possiamo identificare l'evento e cercare di limitare i danni locali o anche evitarli a priori

**Avoidance o azioni di Recovery dopo la identificazione**

# TCP: CONGESTIONE

---

Il caso di congestione è considerato distruttivo per TCP

## Identificazione della congestione

- time out che **scatta in modo ripetuto**: si assume che il pari non sia raggiungibile e che la congestione sia in atto (anche solo 1 timeout!)

**Per recovery, si devono attuare azioni locali per evitare di aggravare il problema e di scongiurare la congestione**

- in caso di congestione, in modo unilaterale, **il mittente dimezza la finestra di invio e raddoppia il time-out**
- al termine **della congestione**, per ritornare ad una situazione di regime si riparte con un transitorio con finestra piccola (**slow start**)

**Slow start è anche la politica iniziale per evitare una potenziale congestione iniziale (avoidance)**

- Le variazioni vengono fatte **in modo dolce** (appunto con uno slow start)
- se mandassimo subito **tutto il flusso**, probabilmente causeremmo dei transitori di congestione su router intermedi

# TCP: CONGESTION CONTROL

---

La connessione TCP adotta meccanismi di controllo della congestione, usando non solo la finestra del ricevente

- Si lavora con una finestra corrente ulteriore, variabile in dimensione, detta di **congestione**, o **congestion window**, **cwnd**
- La finestra segnalata dal ricevente viene considerata a tendere (finestra di controllo di flusso o **receiving window**, **rwnd**, dettata dal ricevente)
- Si considera anche un valore variabile, detto ssthreshold, **ssthresh**, o **soglia di slow start**

Il controllo di congestione lavora in due modi diversi in due fasi distinte di operatività a regime della connessione, anche se con lo stesso obiettivo

- **Slow start** (se  $cwnd < ssthresh$ )
- **Congestion avoidance** (se  $cwnd \geq ssthresh$ )

# TCP: SLOW START

---

Lo **Slow start** è il transitorio sulla finestra del mittente per arrivare da una **situazione iniziale fredda (senza comunicazione)** ad una **comunicazione a regime calda (diversa banda)**

- **rwnd** è il valore a regime, e ci si arriva partendo da finestre molto limitate che crescono in base all'assorbimento della rete e agli ack ricevuti con crescite differenziate
- veloci inizialmente (sotto **ssthreshold**)
  - **fase esponenziale**
- e più limitate successivamente (sopra **ssthreshold**)
  - **fase lineare**

Lo slow start caratterizza la variazione tipica della fase iniziale  
E anche altre variazioni di regime

# TCP: SLOW START

---

**Lo Slow Start va intesa come strategia per passare in modo ben raccordato tra due situazioni**

Ad esempio ogni passaggio da uno stato con un certo regime (anche nullo) ad uno stato successivo con altri parametri usa slow start o passaggio dolce

- Sotto la soglia di slow start si lavora in modo esponenziale e rapido
- Sopra la soglia di slow start si lavora in modo lineare, aggiungendo un segmento e togliendo un segmento alla volta
  - La scelta di due diversi momenti tende ad adeguarsi alla prima fase di passaggio un cui possiamo andare veloci, e anche alla seconda fase in cui dobbiamo fare le cose in modo più lento ed esplorativo

**La strategia slow start caratterizza ogni situazione di variazione, in particolare la fase iniziale, ma anche i casi di congestione e ripartenza, che possono essere molto critici**

# SLOW START - ESEMPIO

---

**Nella fase iniziale si lavora con cwnd (e non rwnd)**

- Si parte con **cwnd = 1 segmento medio**  
Tipicamente di dimensione MSS

Per ogni acknowledgement ricevuto, si incrementa la cwnd in due diversi modi (**esponenziale prima e poi lineare**)

- Tipicamente si raddoppia la dimensione del cwnd, se rimane sotto una soglia (ssthresh)  
Questo consente un incremento veloce (esponenziale), fino ad una certa parte della rwnd
- Tipicamente il cwnd cresce in modo lineare (di 1 segmento), se va oltre la soglia  
Questo consente un incremento più lento e mirato (lineare), fino a arrivare al valore corrente di rwnd

# ANDAMENTO SLOW START TIPICO

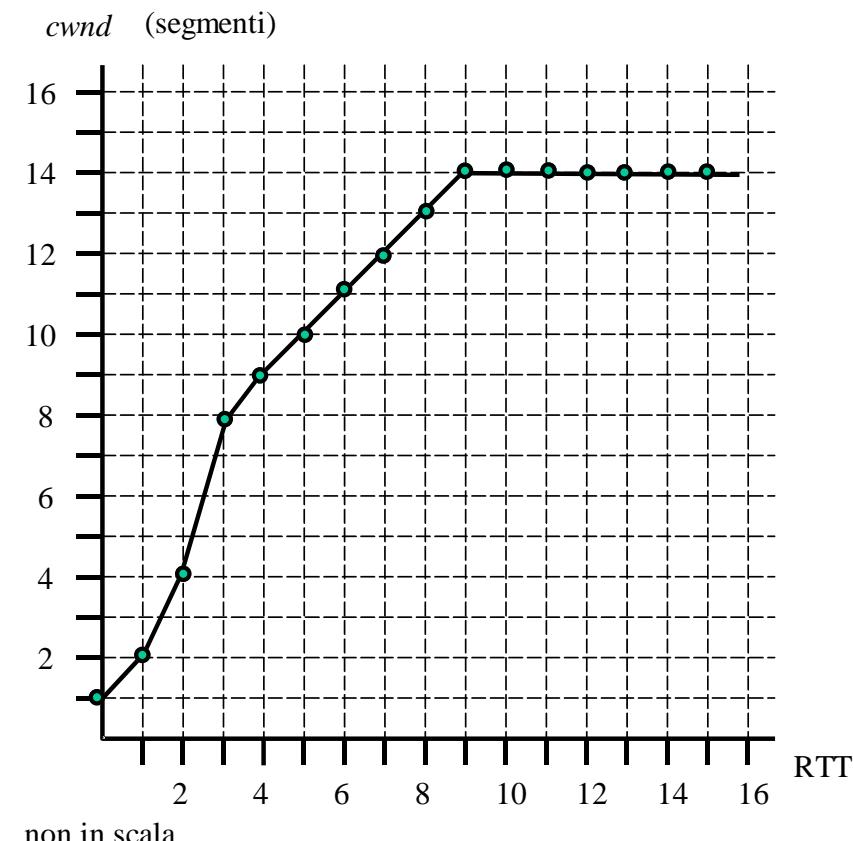
**Slowstart** in fase iniziale o dopo un timeout ripetuto (indicatore di congestione)

**cwnd** finestra corrente o window congestione in numero segmenti

**rwnd** finestra controllo flusso del ricevente

**ssthresh** slow start threshold soglia di base in memoria (64K)

- inizio fase di slow start  
 $cwnd = 1$  segmento,  $ssthresh = 64K$
- per ogni ACK ricevuto correttamente  
if  $cwnd < ssthresh$   
raddoppia  $cwnd$  (fase esponenziale)  
else  $cwnd = cwnd + 1$  (fase lineare)



# CONGESTION AVOIDANCE

In caso di congestione presunta, si ridimensiona tutto e si riparte in modo esplorativo

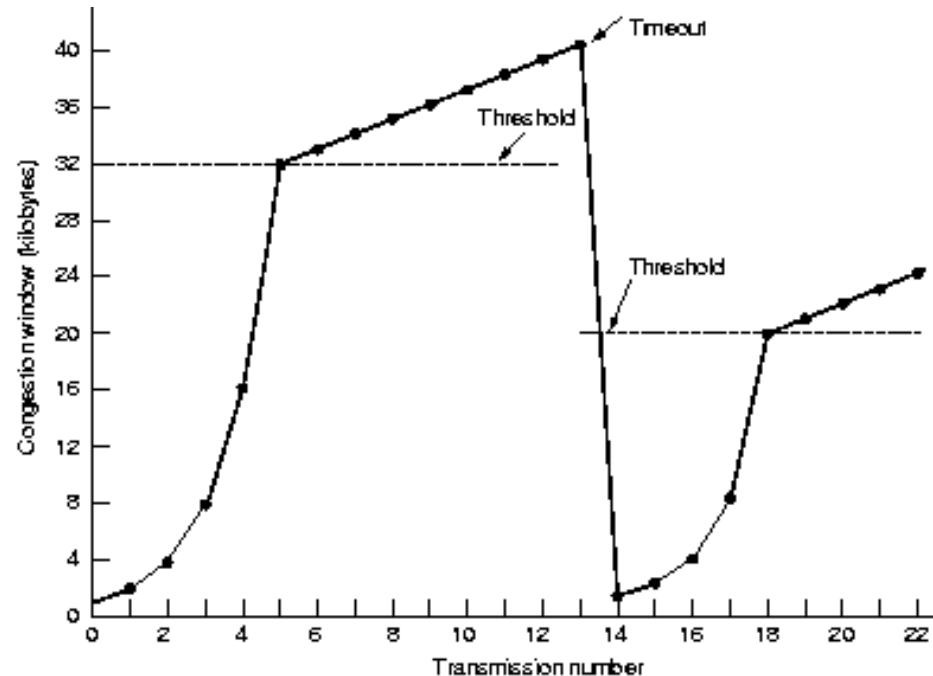
dopo un timeout ripetuto, inteso come indicatore di congestione (o anche un solo timeout e identificazione di congestione)

In caso di evento rilevato di congestione

- si riparte con una congestion window iniziale  
**cwnd = 1**
- si considera una nuova soglia di congestione limitata  
**ssthresh = cwnd/2**
- si riparte con uno slow start

Fino a nuova fase

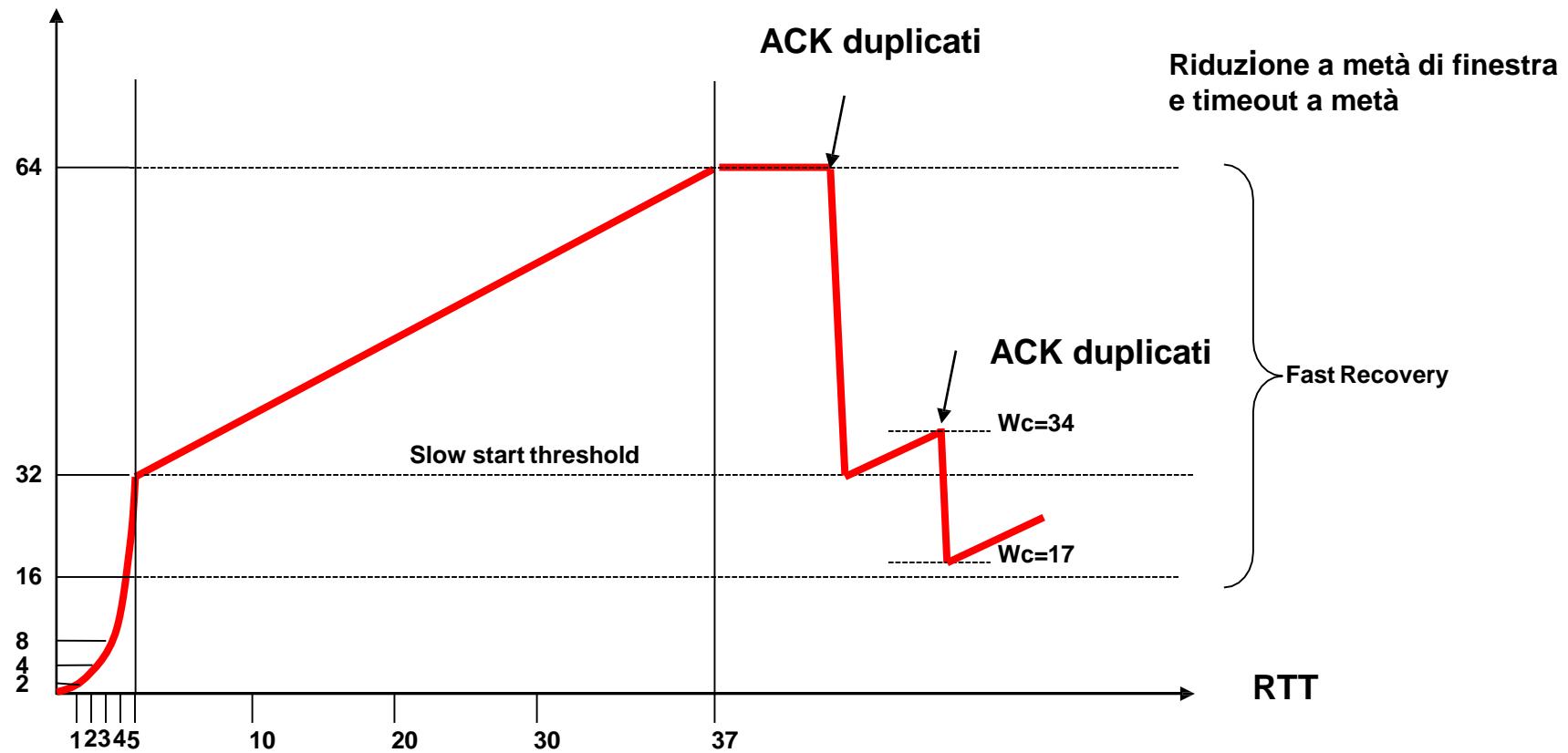
...



# CONGESTION AVOIDANCE

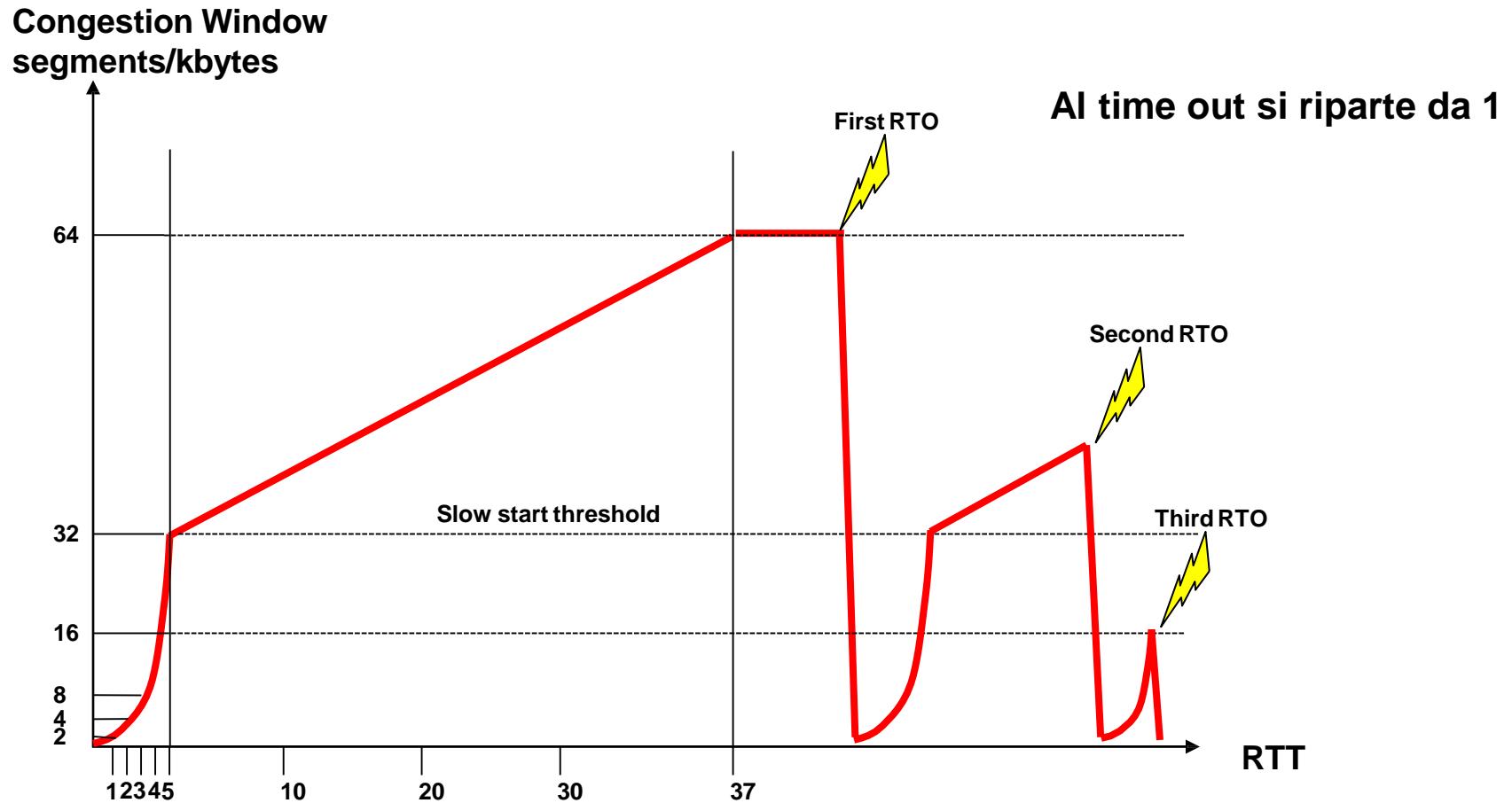
In caso di ack ripetuti, si dimezza la finestra e si raddoppia il timeout

Congestion Window  
segments/kbytes



# CONGESTION AVOIDANCE

In caso di timeout ripetuti, si riparte con slow start



# TIPICA COMUNICAZIONE TCP

---

Uno scenario di uso tipico prevede una fase iniziale in cui i pari si scambiano informazioni su:

- numeri di sequenza iniziali per i flussi
- finestra ricezione
- dimensione media del segmento da scambiare (Maximum Segment Size)
- time-out iniziali da calcolare (vedi la fase iniziale a tre vie)
- opzioni come fattore di scala finestra o altri
- opzioni e estensioni che si possono scambiare: per transazioni ...

Il protocollo prevede un transitorio per arrivare alla situazione di regime slow start

- Si inizia con un segmento nella finestra di congestione, e si raddoppia (exponential backoff) appena arriva un ack; quando la finestra di congestione raggiunge quella di ricezione, siamo a regime e si incrementa/ decrementa di unità alla volta (fase lineare) fino ad eventuali situazioni di congestione

# STRATEGIE TIPICHE IN TCP

---

## ricalcolo del time-out in modo dinamico

- il time out corrente viene tarato rispetto a quanto calcolato come media con la stima del time-out precedente

## exponential backoff

- in caso di ritrasmissione, il time-out raddoppia, dopo raddoppia ancora, fino ad un tempo massimo (ad es. 4'), poi si chiude la connessione

## silly window

- per evitare di lavorare un byte alla volta, non si annunciano finestre di dimensione troppo piccole ( $MSS/2$ ) a parte la finestra chiusa (0 per blocca trasmissioni pari)

## limiti al time-wait

- per limitare la durata delle risorse per la connessione
- Ricordiamo che la memoria sulla porta dovrebbe essere mantenuta per tempi necessari per smaltire tutto il contenuto del buffer ma non troppo superiori a quelli

## long fat pipes

- per mantenere piene le pipe a banda elevata (fornendo indicazioni di buffer superiori a quelli di utente e bufferizzando a livello di supporto)